MAY 2009

virus BULLETIN

Fighting malware and spam

CONTENTS

ISSN 1749-7027

IN THIS ISSUE

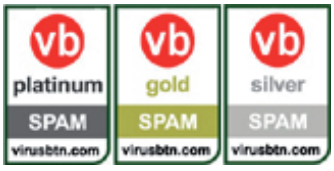
GOING MODULAR

The TDSS modular downloader is known for its ability to bypass active protection, for its outstanding persistence and its rootkit functions. Alisa Shevchenko presents a detailed case study.
page 10

DRAGON'S DEN

Comodo's *Internet Security* suite leaves an overall favourable impression on the VB test team after its first outing on the test bench.
page 15

THE RESULTS ARE IN!



After months of consideration, internal and external discussion, trials and retrials, the results of VB's first 'live' anti-spam comparative review and certification are in. Martijn Grooten has all the details.
page S5

vbSpam supplement

This month: anti-spam news and events; John Levine discusses the ways in which a DKIM-authenticated domain fits into a mail-handling system; and Martijn Grooten reveals the results of *VB*'s first 'live' anti-spam comparative review.

'The most troublesome messages are those that occupy the awkward grey area between ham and spam.'

Helen Martin, Virus Bulletin

ONE MAN'S SPAM

Since the advent of *VB*'s anti-spam testing (see p.S5), each of the *VB* staff members has been tasked with classifying their own incoming mail on a daily basis. This may sound tedious, but I have found that it is not such an onerous task when performed at the start of the day while sipping one's morning coffee and gradually coming to.

It is also a task that concentrates the mind – I find myself taking more time to consider the subject and/or content of many of the messages. While picking out the 'definite' ham from the list displayed in the easy-to-view web interface is extremely easy (by recognizing emails received in one's inbox the previous day), and picking out the 'definite' spam is a no-brainer (e.g. messages in foreign characters, offers from online pharmacies, notifications of lottery wins or suggestions one adds an academic qualification to one's CV), other messages present more of a philosophical challenge.

First, there are the messages that are unwanted ham. For me, these include messages from online retailers from which I have previously made purchases – *Amazon* is a prime example, as are the various companies from which *Virus Bulletin* has purchased marketing materials or office equipment. I am generally happy for these companies to send me information about their latest offers, but nine times out of ten the delete button is applied to the message before it has even been opened. The same is true for numerous newsletters that I have signed up for, as well as alerts from social networking sites. All of these messages go straight into my 'deleted items' folder without even a glance at their content, yet when classifying them I am forced to admit that they are ham. I wonder whether this is an entirely fair classification.

Next, there is the curious phenomenon of spam that is of interest. I hesitate to admit to this, but occasionally there appear messages in my inbox that I know should not be there, but which pique my interest. One recent example was a message from a UK charity promoting a challenge that involves climbing the three highest mountains in the UK within 24 hours. This is not what I would call a classic example of spam, yet I have not signed up to receive messages from the charity in question, nor passed them my email address in any other way. Having absorbed the full details of the message (and decided my levels of fitness are not yet up to the challenge) I reluctantly marked it as spam.

The most troublesome category of messages – that requires the most thought – are those that occupy the awkward grey area between ham and spam. While we have available an 'unclassified' category for messages for which we really are unable to make a decision (for example, for messages sent to a predecessor's email address where we cannot be sure whether or not they signed up to receive them), the use of this category as an easy way out of a tough decision is discouraged.

Being in the publishing business, I find myself at the receiving end of many press releases. PR agencies have found my email address through a variety of sources and send along information which they think will be of interest to me/my publication. Sometimes they get it right, and I receive the latest product news from the players in the IT security industry – these are not messages I have asked for or subscribed to, yet they are certainly of interest. However, others misinterpret the name 'Virus Bulletin' and send me releases on the latest advances in immunology or invite me to biomedical seminars, and yet others let me know about topics as diverse as the launch of a new website for a company that supplies alloy wheels, to the publication of a new book 'for the hard nosed business person to do good in the world AND make a profit'. Beyond the general amusement of reading such announcements I am not interested in them and as such would classify them as spam – but what really makes them any different from those press releases that happen to fall into my subject area of interest? One editor's spam is another's ham.

Finally, I have learned that even spam can bring a smile to an otherwise dreary Monday morning: a recent message arrived in my inbox with the subject line 'We are too lazy to change subjects every daay, please buy our products' [*sic*]. Now that's honesty!

NEWS

RUSSIAN STATE AWARD FOR KASPERSKY

VB extends warm wishes and congratulations to Eugene Kaspersky on being awarded the State Prize of the Russian Federation for Science and Technology.

The prize – the highest Russian award conferred on individuals for services to society and the state – is awarded annually by the President of the Russian Federation to Russian citizens who have demonstrated outstanding work, discoveries and achievements that are deemed to enrich both Russian and global science and that have made significant contributions to the advancement of science and technology.

The CEO and co-founder of *Kaspersky Lab* will be presented with the award by President Dmitry Medvedev at the Kremlin in June.

TREND MAKES NEW ACQUISITION

Demonstrating that businesses can continue to develop and build on their assets in tougher economic climates, *Trend Micro* has announced its acquisition of Canadian security and compliance firm *Third Brigade*. *Trend* CEO Eva Chen claimed that the acquisition would help accelerate ongoing efforts within the company to provide innovative solutions designed specifically for dynamic datacentres. The terms of the agreement have not been disclosed, but the acquisition is expected to be completed in June.

THE EARLY BIRD

Online registration is now open for the VB2009 conference in Geneva this September. Delegates who register before 15 June 2009 will benefit from early bird discounts on the subscriber and non-subscriber rates. VB2009 takes place 23–25 September 2009. The full programme, including abstracts for each paper, can be viewed at http://www.virusbtn.com/conference/vb2009/.

ARCHIVE MATERIAL

Anti-malware products suffered a plague of archive processing vulnerabilities last month, with products from six companies found to be affected. Products from *Avira*, *Aladdin*, *Comodo*, *ESET*, *Trend Micro* and *McAfee* all experienced problems processing archives (which could have led to the scanners failing to detect malicious files contained within an archive). According to *The H Security*, both *Avira* and *ESET* have released an update which has resolved the problem for CAB files; *Comodo* has released an update to fix the bug when processing RAR archives; and *McAfee* has released a fix for the problem with RAR and ZIP archives. Updates are expected soon from *Aladdin* and *Trend Micro*.

Prevalence Table – March 2009

Malware	Type	%
NetSky	Worm	15.65%
Autorun	Worm	12.67%
Mytob	Worm	10.37%
Inject	Trojan	10.24%
Virut	Virus	9.31%
Agent	Trojan	9.12%
Buzus	Trojan	3.84%
Mydoom	Worm	3.80%
Iframe	Exploit	3.02%
Basine	Trojan	2.11%
Downloader-misc	Trojan	2.05%
Delf	Trojan	1.97%
Zafi	Worm	1.78%
Backdoor-misc	Trojan	1.68%
Bagle	Worm	1.64%
Invoice	Trojan	1.31%
Banload	Trojan	1.17%
Suspect packers	Misc	0.83%
Zlob/Tibs	Trojan	0.55%
Sality	Virus	0.53%
Murlo	Trojan	0.51%
Small	Trojan	0.46%
Tenga	Worm	0.41%
Parite	Worm	0.41%
LDPinch	Trojan	0.39%
OnlineGames	Trojan	0.31%
Brontok/Rontokbro	Worm	0.31%
Fuzen	Rootkit	0.28%
Cutwail/Pandex/Pushdo	Trojan	0.28%
Alman	Worm	0.26%
Mabutu	Worm	0.26%
Bifrose/Pakes	Trojan	0.25%
VB	Worm	0.24%
Others[1]		2.94%
Total		100.00%

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

TECHNICAL FEATURE

ANTI-UNPACKER TRICKS – PART SIX

Peter Ferrie
Microsoft, USA

New anti-unpacking tricks continue to be developed as the older ones are constantly being defeated. This series of articles (see also [1–5]) describes some tricks that might become common in the future, along with some countermeasures.

This article concentrates on anti-debugging tricks that target plug-ins for the *OllyDbg* debugger. All of the techniques described here were discovered and developed by the author.

OllyDbg plug-ins

OllyDbg is perhaps the most popular user-mode debugger. A number of packers have been written that are able to detect *OllyDbg*, so plug-ins have been created to attempt to hide it from those packers.

Last month we looked at *antiAnti*, *HideDebugger*, *HideOD*, *IsDebugPresent*, *Olly Advanced* and *OllyICE*. In this article we look at some more *OllyDbg* plug-ins and the vulnerabilities that could be used to detect them.

Olly Invisible

Olly Invisible hooks the code in *OllyDbg* that is reached when it is formatting the kernel32 OutputDebugStringA() string, and then attempts to replace all '%' characters with ' ' in the message. However, a bug in the routine causes it to miss the last character in the string.

The plug-in hooks the debuggee's kernel32 OutputDebugStringA() function by replacing the first six bytes with an indirect jump to a dynamically allocated block of memory. This block attempts to replace all '%' characters with '_' in the message.

Similarly, *Olly Invisible* hooks the debuggee's kernel32 OutputDebugStringW() function by replacing the first six bytes with an indirect jump to a dynamically allocated block of memory. This block attempts to replace all '%' characters with '_' in the message.

The plug-in hooks the debuggee's kernel32 IsDebuggerPresent() function in the same way – by replacing the first six bytes of the function with an indirect jump to a dynamically allocated block of memory. In this case the block always returns zero, regardless of the value in the PEB->BeingDebugged flag.

Olly Invisible hooks the debuggee's ntdll NtQueryInformationProcess() function in the same way again, replacing the first six bytes of the function with an indirect jump to a dynamically allocated block of memory. This block calls the original ntdll NtQueryInformationProcess() function, and then checks whether an error occurred. If no error occurred, then the block checks if the ProcessInformationClass is the ProcessDebugPort class, and that the ProcessInformation parameter is non-zero, and then checks that four bytes are writable at the specified memory address. If all of these requirements are met, *Olly Invisible* writes a zero to the memory address at which the ProcessInformation parameter points. This method is almost unflawed, but it omits a check of whether the current process is specified. However, the current process can be specified in ways other than the pseudo-handle that is returned by the kernel32 GetCurrentProcess() function, and that must be taken into account.

If possible, *Olly Invisible* patches the debuggee's ntdll CsrGetProcessId() function, so that it always returns zero. However, since this function should never return zero, such a result is a sure sign that the plug-in is present.

Olly Invisible hooks the debuggee's ntdll NtQuerySystemInformation() function by replacing the first six bytes with an indirect jump to a dynamically allocated block of memory. This block calls the original ntdll NtQuerySystemInformation() function, and then checks if an error occurred. If no error occurred, it checks if the SystemInformationClass is the SystemProcessInformation class. If it is, then the block searches within the returned process list for processes with the image name 'OllyDbg.exe'. If any are found, then the block adjusts the list so that it skips those entries. However, the entries themselves are untouched, and can be found by a brute-force search of the returned buffer.

Olly Invisible hooks the debuggee's ntdll NtReadVirtualMemory() function by replacing the first six bytes of the function with an indirect jump to a dynamically allocated block of memory. This block calls the original ntdll NtReadVirtualMemory() function, and then checks if an error occurred. If no error occurred, it checks if the read includes the address of a hooked function. If it does, then the block restores the original bytes of the function in the returned buffer, thus achieving in-memory stealth for remote processes. However, there are three problems in the code.

The first problem is in the bounds check: *Olly Invisible* only checks if the read includes the address of the first byte of a hooked function. This means that if the read begins one byte after the start of the hooked function, then the hook will be visible. The second problem is that *Olly Invisible* does not check how many bytes have been read, but always attempts to restore the six altered bytes. Thus, even if only one byte was read, six bytes will be written to the buffer. If the buffer

is at the end of a page or in a sensitive location, then an exception or memory corruption could occur as a result. The third problem is that *Olly Invisible* does not check the process handle for which the request was made, which can lead to the 'stealthing' of the memory of a completely different process. The correct behaviour would be to restore the bytes only if the current process is specified. However, the current process can be specified in ways other than the pseudo-handle that is returned by the kernel32 GetCurrentProcess() function, and that must be taken into account.

Olly Invisible sets the debuggee's PEB->BeingDebugged flag to zero.

The author of *Olly Invisible* has not responded to the report.

PhantOm

The *PhantOm* plug-in changes the 'OllyDbg - <filename> - [CPU]' string in *OllyDbg* to 'PhantOm - [CPU]'.

It changes the 'CPU -' string either to the one specified in the phantom.ini file, or to 'o_O' if no string is specified. It changes the '%smodule' string to '%sm0dule', and changes the 'NULL thread' string to 'NULL thr3ad'.

PhantOm changes the export address for the debuggee's ntdll NtQueryInformationProcess() function so that it points into the file header of ntdll.dll. It also changes the corresponding import address in the debuggee's kernel32.dll to point into the file header of ntdll.dll. It copies the original ntdll NtQueryInformationProcess() function code into the file header of ntdll.dll, and then appends some code to the copied function. The appended code checks the ProcessInformationClass parameter. If the ProcessTimes class is specified, then the hook returns an error. The purpose of the change to kernel32.dll is to hook the kernel32 GetProcessTimes() function implicitly. The ProcessTimes class can be used to expose the length of time that a user requires to debug an application, so by hiding this information, it hides *OllyDbg* too.

PhantOm aims to patch the debuggee's user32 BlockInput() function code to always return successfully, but this code does not work in *Windows 2000* and earlier because of an apparently reversed conditional statement.

PhantOm erases the dwX, dwY, dwXSize, dwYSize, dwXCountChars, dwYCountChars and dwFillAttribute fields from the RTL_USER_PROCESS_PARAMETERS block. These characteristics are checked by the ChupaChu debugger test, which also checks whether bit 7 is set in the dwFlags field. However, due to a bug, the latter check always fails. If it were not for the bug, the ChupaChu test would detect the plug-in.

PhantOm attempts to hook the debuggee's ntdll KiUserExceptionDispatcher() function by replacing an 0xE8 opcode ('CALL' instruction) with an 0xE9 opcode ('JMP' instruction) at a fixed location within the routine. This behaviour is a bug, because in *Windows Vista* the routine has an additional instruction prepended to it, meaning that the required instruction is in a different location. However, if the hook is successful, then when an exception occurs, the hook saves the state of the debug registers into a private memory region. The hook then swaps in the previous debug register values before passing the exception to the debuggee. This tricks the debuggee into thinking that any changes it makes are current. *PhantOm* also attempts to hook the ntdll NtContinue() function in order to save the updated debug register values on return from the debuggee. However, a bug exists in the hooking code. The hook checks for the correct instruction before replacing it, but due to an incorrect conditional assignment, it performs the replacement regardless of the result.

PhantOm hooks the debuggee's kernel32 GetTickCount() function by replacing the first five bytes of the function with a relative jump to a dynamically allocated block of memory. This block intercepts attempts to call the kernel32 GetTickCount() function, and then returns a tick count that is incremented by one each time it is called, regardless of how much time has passed.

PhantOm patches __fuistq() in *OllyDbg* to avoid the floating-point operations error. It does this by skipping the data conversion. This is not the proper way to avoid the problem, however, since no values are converted as a result. A better fix would be to change the floating-point exception mask to ignore such errors. This can be achieved by changing the dword at file offset 0xCB338 from 0x1332 to 0x1333, or just by loading that value manually into the control word of the FPU.

PhantOm patches the code in *OllyDbg* that is reached when it is formatting the kernel32 OutputDebugStringA() string. The patch prevents the debugger from formatting the message.

PhantOm hooks the code in *OllyDbg* that is reached when a debug event occurs. When the hook is reached, it checks for the following events:

- If the DBG_PRINTEXCEPTION_C (0x40010006) exception is seen, then the hook returns a status that the exception was not handled. This hides *OllyDbg* from the kernel32 GetLastError() detection method.
- If the EXCEPTION_ACCESS_VIOLATION (0xC0000005) or EXCEPTION_GUARD_PAGE (0x80000001) exception is seen and is not within the bounds of a memory breakpoint, then the hook returns a status that the event was not handled. This hides *OllyDbg* from the guard page detection method.

- If the EXCEPTION_ ILLEGAL_INSTRUCTION (0xC000001D), EXCEPTION_INVALID_LOCK_ SEQUENCE (0xC000001E) or EXCEPTION_ INTEGER_DIVIDE_BY_ZERO (0xC00000094) exception is seen, then *PhantOm* returns a status that the event was not handled. This prevents *OllyDbg* from breaking on several common conditions.

PhantOm installs a driver which hooks the NtQueryInformationProcess(), NtOpenProcess(), NtClose(), NtSetInformationThread(), NtYieldExecution(), NtQueryObject(), NtQuerySystemInformation() and NtSetContextThread() functions in ntoskrnl.exe by name, and the GetWindowThreadProcessId(), EnumWindows(), FindWindowA() and GetForegroundWindow() functions in ntoskrnl.exe by service table index. What happens next depends on the hook that is called:

- When the NtQueryInformationProcess() function is called, the hook checks the ProcessInformationClass parameter. If the ProcessDebugPort class was specified, then the hook zeroes the debug port, but without checking the process handle. The correct behaviour would be to zero the port only if the current process is specified. However, the current process can be specified in ways other than the pseudo-handle that is returned by the kernel32 GetCurrentProcess() function, and that must be taken into account.

If the ProcessBasicInformation class was specified, then the hook replaces the process ID of *OllyDbg* with the process ID of EXPLORER.EXE in the InheritedFromUniqueProcessId field. This could be considered a bug, since the true parent might not be *Explorer*. The proper behaviour would be to use the process ID of *OllyDbg*'s parent.

- When the NtOpenProcess() function is called, the hook checks if the process ID to open matches the process ID of *OllyDbg* or CSRSS.EXE, and returns an error in the latter case.
- When the NtClose() function is called, the hook checks for a valid handle before attempting the close. This hides *OllyDbg* from the CloseHandle(invalid) detection method.
- When the NtSetInformationThread() function is called, the hook checks if the HideThreadFromDebugger class has been specified, and returns success if that is the case. There is a bug in this code, which is that if an invalid handle is passed to the function, then an error code should be returned. A successful return would be an indication that *PhantOm* is running.
- When the NtYieldExecution() function is called, the hook always returns a status. This hides *OllyDbg* from the NtYieldExecution() detection method.
- When the NtQueryObject() function is called, the hook checks for the ObjectAllTypesInformation class, and then erases all the returned information if it is specified.
- When the NtQuerySystemInformation() function is called, the hook checks the SystemInformationClass parameter. If the SystemKernelDebuggerInformation class is specified, then the hook erases all of the returned information. If the SystemProcessInformation class is specified, then the hook adjusts the list to skip those entries. However, the entries are untouched and can be found by a brute-force search of the returned buffer.
- When the NtSetContextThread() function is called, the hook clears the CONTEXT_DEBUG_REGISTERS flag from the ContextFlags field before completing the call. This prevents the debug register values from being returned, and hides *OllyDbg* from the debug registers detection method.
- When the GetWindowThreadProcessId() function is called, the hook checks whether the process ID matches the process ID of *OllyDbg*, and returns zero if that is the case. This technique hides *OllyDbg* from the window handle detection method.
- When the EnumWindows() function is called, the hook removes from the list all windows whose process ID matches that of *OllyDbg*. This technique hides *OllyDbg* from the window handle detection method.
- When the FindWindow() function is called, the hook checks whether the returned window handle belongs to *OllyDbg*, and returns zero if that is the case.
- When the GetForegroundWindow() function is called, the hook checks whether the returned window handle belongs to *OllyDbg*, and returns the previous foreground window handle in that case.

PhantOm installs a driver that makes the RDTSC instruction illegal when called from ring 3. The driver intercepts the exception that occurs when the instruction is issued. When the exception occurs, the driver executes the RDTSC instruction in ring 0, and then uses the low byte of the returned value as the time elapsed since the last time the RDTSC instruction was executed. This has the effect of slowing perceived time, and hides *OllyDbg* from the RDTSC detection method.

PhantOm sets the debuggee's PEB->BeingDebugged flag to zero.

One of the authors of *PhantOm* responded to the report: the BlockInput() bug will be fixed in a future version; the KiUserExceptionDispatcher() and NtContinue() bugs will remain, because *Windows Vista* is not supported.

Stealth64

The *Stealth64* plug-in forces *OllyDbg* to ignore the OptionalHeader bug described in [6].

Stealth64 patches the code in *OllyDbg* that is reached when it reads the debuggee's imported function names. The patch stops *OllyDbg* from displaying an error message if an imported function name cannot be read.

The plug-in patches the code in *OllyDbg* that is reached when it parses the debuggee's Import Table. The patch stops *OllyDbg* from displaying an error message if the import table appears to be corrupted.

Stealth64 patches the code in *OllyDbg* that is reached when it parses the debuggee's Base Relocation Table. The patch stops *OllyDbg* from applying relocations. However, this also prevents *OllyDbg* from debugging certain files.

Stealth64 handles the exception-priority trick described in [1] by forcing a single-step exception to occur in the ntdll KiUserExceptionDispatcher() function.

The plug-in sets the debuggee's PEB->BeingDebugged and PEB->NtGlobalFlag flags to zero.

Stealth64 hooks the debugger's kernel32 CreateProcessA() function. The hook defines and sets the '_NO_DEBUG_HEAP' environment variable to one, before calling directly into the kernel32 CreateProcessInternalA() function. This environment variable forces a process to use a standard heap instead of a debugging heap, even if the process is being debugged.

Stealth64 removes the SeDebugPrivilege from the process token.

Stealth64 hooks the debuggee's ntdll KiUserExceptionDispatcher() function. When an exception occurs, the hook saves the state of the debug registers into a private memory region if the 'ProtectDRX' option is enabled. The hook swaps in the previous debug register values if the 'HideDRX' option is enabled, before passing the exception to the debuggee. This tricks the debuggee into thinking that any changes it makes are current. *Stealth64* also hooks the ntdll NtContinue() function, in order either to save the updated debug register values on return from the debuggee if the 'HideDRX' option is enabled, or to swap back the original debug register values if the 'ProtectDRX' option is enabled.

Stealth64 searches within up to 256 bytes of the debugger's ntdll DbgUiConvertStateChangeStructure() function for a reference to the DBG_PRINTEXCEPTION_C (0x40010006) exception, followed by an 0x75 opcode ('JNE' instruction). If the sequence is found, then it replaces the 0x75 opcode with an 0xEB opcode ('JMP' instruction). The ntdll DbgUiConvertStateChangeStructure() function was introduced in *Windows XP*, but *Stealth64* runs only in *Windows Vista64*, so there is no problem with earlier versions of *Windows*. The effect of the patch is to prevent the OUTPUT_DEBUG_STRING_EVENT debug event from being delivered to the debugger. Instead, a generic EXCEPTION_DEBUG_EVENT debug event is delivered to the debugger. This hides *OllyDbg* from the GetLastError() detection method. However, there is a bug in the search routine, which assumes that all five bytes can be read. If the read accesses out-of-bounds memory, then *OllyDbg* will crash.

Stealth64 intercepts the EXCEPTION_GUARD_PAGE (0x80000001) exception and checks the address at which the fault occurred. If the fault is not within the bounds of a memory breakpoint, then the hook returns a status that the event was not handled. This hides *OllyDbg* from the guard page detection method.

Stealth64 changes the address in each of the debuggee thread's TEB->Wow32Reserved field values, to point to a dynamically allocated block of memory. That field is undocumented, but it normally points into a function within the wow64cpu.dll which orders the parameters for a 64-bit system call, and then falls into the wow64cpu TurboDispatchJumpAddressStart() function to perform the transition to kernel mode. By changing this field value, *Stealth64* creates a clean single point of interception for all system calls.

The block that *Stealth64* allocates contains code to watch for particular system table indexes. This act ties *Stealth64* to a specific version of *Windows Vista64*. The indexes that are intercepted are: NtQueryInformationProcess, NtQuerySystemInformation, NtSetInformationThread, NtClose, NtOpenProcess, NtQueryObject, FindWindow, BlockInput, NtQueryPerformanceCounter, BuildHwndList, NtProtectVirtualMemory and NtQueryVirtualMemory. If none of these indexes is seen, and if the 'HandleSingleStepExceptions' option is enabled, then *Stealth64* will register a Vectored Exception Handler. That handler consumes EXCEPTION_SINGLE_STEP (0x80000004) exceptions that occur in the region of memory that includes the injected code.

If the NtQueryInformationProcess index is seen, then the hook calls the original TEB->Wow32Reserved pointer and checks if the function has succeeded. If it has, then the hook checks the ProcessInformationClass parameter. If the ProcessDebugPort class is specified, then the hook zeroes the port and returns success. If the ProcessDebugObjectHandle class is specified, then the hook zeroes the handle and returns STATUS_PORT_NOT_SET (0xC0000353). If the ProcessDebugFlags class is specified, then the hook sets the flags to true, signifying that no debugger is present, and returns success. The correct behaviour for these three classes

is for the changes to be applied only if the current process is specified. However, the current process can be specified in ways other than the pseudo-handle that is returned by the kernel32 GetCurrentProcess() function, and that must be taken into account.

If the ProcessBasicInformation class is specified, then the hook replaces the process ID of *OllyDbg* with the process ID of EXPLORER.EXE in the InheritedFromUniqueProcessId field. This could be considered a bug, since the true parent might not be *Explorer*. The proper behaviour would be to use the process ID of *OllyDbg*'s parent.

If the NtQuerySystemInformation index is seen, the ReturnLength is zero and the SystemInformationClass is the SystemProcessInformation class, then the hook uses the TIB->ArbitraryDataSlot field to hold the returned length. There is a bug here, which is that the previous value in that field is not saved, and the hook always zeroes it before returning. The problem with this approach is that it can be detected by malware that sets the TIB->ArbitraryDataSlot field value to non-zero, then calls the ntdll NtQuerySystemInformation() function with no ReturnLength parameter. *Stealth64* is revealed because the TIB->ArbitraryDataSlot field value is zero.

In any case, the hook calls the original TEB->Wow32Reserved pointer, and then checks if the function has succeeded. If it has, then the hook checks that the 'Fake Parent' option is enabled. If it is, then the hook replaces the process ID of *OllyDbg* with the process ID of EXPLORER.EXE in the InheritedFromUniqueProcessId field. This could be considered another bug, since the true parent might not be *Explorer* (as before, the proper behaviour would be to use the process ID of *OllyDbg*'s parent).

The hook also checks if the 'NtQuerySystemInformation' option is enabled. If it is, then the hook parses the returned process list. The hook deletes the entry that corresponds to *OllyDbg* by copying the entries that follow over the top and then reducing the returned length.

If the NtSetInformationThread index is seen, and the ThreadInformationClass is the HideThreadFromDebugger class, then the hook returns success. There is a bug in this code, which is that if an invalid handle is passed to the function, then an error code should be returned. A successful return would be an indication that *Stealth64* is running.

If the NtClose index is seen, then the hook calls the ntdll NtQueryObject() function to verify that the handle is valid. If it is, then the hook calls the ntdll NtClose() function. Otherwise, it returns STATUS_INVALID_HANDLE (0xC0000008).

If the NtOpenProcess index is seen, then the hook attempts to replace the process ID of *OllyDbg* with the process ID of EXPLORER.EXE in the address to which the ClientId parameter points. However, there are three bugs here: the first is that the hook does not check if the ClientId parameter points to a valid memory location. An invalid memory address causes an exception that can be intercepted by the debuggee. Such an exception is a sure sign that *Stealth64* is running. The second bug is that the hook does not check if the ClientId parameter points to a writable memory location prior to attempting to replace the process ID. Writing to a read-only memory address causes an exception that can be intercepted by the debuggee. Such an exception is an indication that *Stealth64* is running. The third bug is that the hook zeroes the upper 32 bits of the quadword to which the ClientId parameter points. This can allow the function to succeed in places where it should fail. A successful return in that case is another sign that *Stealth64* is running.

If the NtQueryObject index is seen, then the hook calls the original TEB->Wow32Reserved pointer, then checks if the function has succeeded. If it has, then the hook checks if the ObjectInformationClass is the ObjectAllTypesInformation class. If it is, then the hook searches the returned buffer for all objects whose length is 0x16 bytes, and then zeroes the object counts, without checking the object name. This is a bug, since there could be other objects with the same name length, and their handle counts will also be zeroed.

If the FindWindow index is seen, then the hook calls the original TEB->Wow32Reserved pointer, and then checks if the function has succeeded. If it has, then the hook calls the user32 GetWindowThreadProcessId() function for the returned window handle. The hook returns zero if the returned process ID matches the process ID of *OllyDbg*.

If the BlockInput index is seen, then the hook simply returns. This behaviour is a bug, since the return code is never set.

If the NtQueryPerformanceCounter index is seen, then the hook calls the original TEB->Wow32Reserved pointer, and then checks if the function has succeeded. If it has, then the hook returns a tick count that is incremented by one each time it is called, regardless of how much time has passed.

If the BuildHwndList index is seen, then the hook calls the original TEB->Wow32Reserved pointer, and then checks if the function has succeeded. If it has, then the hook parses the returned hwnd list and then deletes the entry that corresponds to *OllyDbg* by copying the entries that follow over the top, and then reducing the returned length.

If the NtProtectVirtualMemory index is seen, then the hook checks if the ProcessHandle parameter corresponds to the GetCurrentProcess() pseudo-handle. If it does, then the hook checks if the value in the memory location to which the BaseAddress parameter points matches the location of the internal breakpoint address that *Stealth64* uses. If it

does, then the hook returns success. There are three bugs in this code, and one behaviour that could be considered a bug. The first bug is that the hook does not check if the BaseAddress parameter points to a valid memory location. An invalid memory address causes an exception that can be intercepted by the debuggee. Such an exception is a good indication that *Stealth64* is running. The second bug is that the BaseAddress parameter can span the region that is protected by the internal breakpoint, and as a result the comparison will fail. The third bug is that to return success if the comparison succeeds might be incorrect behaviour if the NewAccessProtection parameter specifies an invalid protection value. In that case, an error code should be returned instead. The behaviour that could be considered a bug is that the ProcessHandle parameter might contain a handle to the current process, as returned by the kernel32 OpenProcess() function. This handle will not be recognized as belonging to the current process.

If the NtQueryVirtualMemory index is seen, then the hook checks if the BaseAddress parameter matches the location of the internal breakpoint address that *Stealth64* uses, and that the VirtualMemoryInformationClass parameter is zero. If those checks succeed, then the hook calls the original TEB->Wow32Reserved pointer, and attempts to set the value in the VirtualMemoryInformation->Protect field to Executable/Readable/Writable, if the VirtualMemoryInformation parameter has been specified.

There are four bugs in this code. The first is that the hook does not check if the function call succeeded. The second bug is that the hook does not check if the VirtualMemoryInformation parameter points to a valid memory location. An invalid memory address causes an exception that the debuggee can intercept. Such an exception is a sure sign that *Stealth64* is running. The third bug is that the hook does not check if the VirtualMemoryInformation parameter points to a writable memory location prior to attempting to write the VirtualMemoryInformation->Protect field value. Writing to a read-only memory address causes an exception that the debuggee can intercept. Such an exception is a sure sign that *Stealth64* is running. The fourth bug is that the hook does not check the process handle for which the request was made, which can lead to lying about the memory state of a completely different process. The correct behaviour would have been to check if the current process is specified. However, the current process can be specified in ways other than the pseudo-handle that is returned by the kernel32 GetCurrentProcess() function, and that must be taken into account.

A HandleInt2D option exists but it is not supported.

The author of *Stealth64* responded to the report, and the bugs will be fixed in a future version.

Olly's Shadow

Olly's Shadow is a patched and renamed version of *OllyDbg*. Since it is renamed, it hides *OllyDbg* from the standard FindWindow() and process enumeration detection techniques. *Olly's Shadow* does not export any functions, which avoids another common detection method on the export name table. However, this prevents the use of plug-ins, unless they use hard-coded addresses.

Olly's Shadow behaves like *Olly Invisible* with respect to the OutputDebugString handling, complete with the same bug: *Olly's Shadow* hooks the code in *OllyDbg* that is reached when *OllyDbg* is formatting the kernel32 OutputDebugStringA() string, and then attempts to replace all '%' characters with ' ' in the message. However, a bug in the routine causes it to miss the last character in the string.

Olly's Shadow changes the options that are used when loading symbols, and then disables the name merging. This avoids several problems with corrupted symbol files, including the dbghelp.dll bug described in [1].

Olly's Shadow changes the class name from 'OLLYDBG' to 'SHADOW', and the window title from 'OllyDbg' to 'Shadow'.

The author of *Olly's Shadow* could not be contacted.

In the final part of this series next month we will look at anti-debugging tricks that target other popular debuggers, as well as some anti-emulating and anti-intercepting tricks.

The text of this paper was produced without reference to any Microsoft source code or personnel.

REFERENCES

[1] Ferrie, P. Anti-unpacker tricks – part one. Virus Bulletin, December 2008, p.4. http://www.virusbtn.com/pdf/magazine/2008/200812.pdf.

[2] Ferrie, P. Anti-unpacker tricks – part two. Virus Bulletin, January 2009, p.4. http://www.virusbtn.com/pdf/magazine/2009/200901.pdf.

[3] Ferrie, P. Anti-unpacker tricks – part three. Virus Bulletin, February 2009, p.4. http://www.virusbtn.com/pdf/magazine/2009/200902.pdf.

[4] Ferrie, P. Anti-unpacker tricks – part four. Virus Bulletin, March 2009, p.4. http://www.virusbtn.com/pdf/magazine/2009/200903.pdf.

[5] Ferrie, P. Anti-unpacker tricks – part five. Virus Bulletin, April 2009, p.4. http://www.virusbtn.com/pdf/magazine/2009/200904.pdf.

[6] Ferrie, P. Anti-unpacker tricks. http://pferrie.tripod.com/papers/unpackers.pdf.

FEATURE

CASE STUDY: THE TDSS ROOTKIT

Alisa Shevchenko
eSage Lab, Russia

This article is a case study of the TDSS malware, also known as Tidserv, TDSServ and Alureon. Some of its components are detected as Trojan.Win32.DNSChanger and Trojan.FakeAlert.

There are several reasons for conducting a detailed study of this malware:

1. Disinfection of TDSS seems to be problematic for modern anti-malware solutions. At the time of writing this article, a *Google* search for the malware [1] results in a considerable number of forum posts from desperate users whose anti-virus solutions have detected the malware, but failed to remove it.
2. Detailed descriptions of this malware are not available publicly.
3. TDSS is not rocket science! Despite being quite advanced and posing problems for anti-malware solutions, it does not engage any outstanding new techniques.
4. TDSS is actively spreading in the wild and developing into a wide and mighty botnet. According to *Kaspersky Lab* [2], between 100 and 300 signature detections are being added per day for new/modified TDSS components.

Thus, TDSS is a borderline type of threat: sufficiently advanced to cause problems for AV, or even to defeat it completely, but not sufficiently critical to trigger a detailed study; widespread enough to cause numerous user issues, but not serious enough to trigger a full epidemic alert.

FAMILY OVERVIEW

TDSS is known for its ability to bypass active protection/HIPS, for its outstanding persistence and its rootkit functions. Users with all kinds of anti-malware solutions have reported problems disinfecting their systems. Observable activity typically includes website redirects, ad popups and the blocking of AV updating/loading activities. Its functionality can vary widely though, since TDSS is designed as a modular unit and additional components can be downloaded and installed to provide extra features.

The first TDSS infection reports date back to the middle of 2008. Even at that time the malware showed extraordinary persistence, causing problems for users and demonstrating the ability to bypass anti-malware protection. Given that the malware's creators have managed to keep this advanced functionality up to date for almost a year now, and given the malware's code architecture and skilful implementation, we can assume that TDSS is being developed with a clear vision by a team of proficient engineers.

TDSS itself is a very advanced modular downloader. Its main goal is to persist in a system and to provide a means for remote control (via a downloaded configuration file) and a framework for downloading/installing modules for additional functionality.

TDSS is delivered to a PC through a wide and elaborate distribution network. Known attack vectors include website iframe attacks [3, 4] and bundling the malware with pseudo-legitimate video codecs [5], as well as legitimate software [6] and cracks [7] distributed via p2p networks.

Family traits

- The original name of TDSS (assigned by its creators) is 'TDL'. The most recent samples call themselves 'TDL2'.
- The trojan files are protected from binary analysis using code obfuscation and encryption.
- Some files contain a fake *Microsoft* version stamp.
- TDSS is installed when the msiexec.exe (*Microsoft Installer*) service loads a legitimate, but maliciously patched DLL [8].
- After installation, the trojan effectively prevents anti-virus software from launching or updating.
- The trojan is persistent through a variety of techniques. For example, some of the family members survive Safe Boot. This is achieved by registering the trojan's driver in the HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Minimal and HKLM\SYSTEM\ControlSet001\Control\SafeBoot\Network registry keys.
- The trojan creates a (hidden) registry key to store its configuration information, such as the AV modules that are to be denied Internet access, and the malicious modules that are to be injected into browsers.
- The trojan hides its files and registry values by means of several system hooks.
- The trojan uses the hooked function ZwFlushInstructionCache as a communication gateway to its own kernel driver.

FAMILY DIVERGENCE & RECENT UPDATES

Back in 2008, the presence of TDSS was marked by a driver named TDSSserv.sys (after which the malware

was named). Since then, malware-related file names have changed several times, and have included clbdriver.sys, seneka*.sys, UACd*.sys, gaopdx*.sys, tdlserv.sys and others.

Another change is that recent samples patch msi.dll for their installation, while the early samples used to patch advapi32.dll. This is probably a reaction to the behavioural heuristics that have recently been added to security solutions.

In the most recent samples the code protection is designed to make the trojan look like a regular system file or a device-supporting utility. The unpacker stub is a big piece of regular code, which means there is no extra entropy throughout the file's byte array (which is an easy-to-spot sign of a packed file). Furthermore, the code is enriched by random pseudo-legitimate ASCII strings and random API calls designed to fool a hasty analyst into thinking it is a legitimate piece of code.

The code protection itself is trivial: an easily removed envelope with normal code inside.

Most recent samples of TDSS contain worm functionality. The malware tries to distribute itself to removable drives by copying its own body into all available drives as a hidden *.com file in the hidden RECYCLER directory, and by creating the file autorun.inf, with the file reference on the same drive.

Most recent TDSS samples change systems' DNS addresses, thus causing all the hostname requests to filter through a malicious service. This is a brilliant solution, probably inspired by the much-talked-about DNS root server vulnerability and the Evilgrade proof of concept [9]. Distributing a spoofed DNS provider throughout the network by means of a DHCP service gives an attacker control of the entire network's web traffic, even as far as delivering malware to clean machines under the guise of a legitimate software update.

SAMPLE ANALYSIS

For analysis, I took a fairly recent sample, dated March/April 2009 (MD5: 1DE66FC07C7B5893F5F83B397AC38F3D). It is a specimen of the TDSS variety quoted by *Symantec* Russia as being one of the most notable at the end of March [10, 11].

The general execution flow of an average TDSS specimen has already been described [9, 12], as have its basic mechanisms in userland [4]. A summary of the high-level functions of this particular sample is available from any public sandbox [13]. I will be focusing on the trojan's most important features and driver functionality.

Trojan installation and protection bypassing

The trojan's initial installation routine is notable, since it allows behavioural protection/firewalls to be bypassed. The idea is to force a legitimate service to load a legitimate, but maliciously patched DLL. This is achieved via the modification of the msi.dll file in the \knowndlls directory, followed by a regular launch of the *Microsoft Installer* service:

```
NtCreateSection(..”\knowndlls\dll.dll”..) // new section for a malicious dll

CopyFile(..”msi.dll”, <temporary_file>..) // preparing the dll to patch

WriteFile(..<temporary_file>, <malicious_code_injection>..) // patching
```

The injected code will call LoadLibrary, which will invoke the malicious dll mapped into the \knowndlls\dll.dll section. The shellcode is quite elegant:

```
push 7c906cbc ; pointer to ‘dll.dll’ – really this is a calculated pointer to the last part of the ‘ntdll.dll’ name in the regularly mapped ntdll.dll

call $+5 ; call next instruction so that its address is on the stack

sub  dword ptr [esp], 0a ; now the first dword on the stack points to the first shellcode instruction, meaning that LoadLibrary will return there. Shellcode will be replaced by original code by then.

mov  eax, LoadLibrary

jmp  eax ; call LoadLibrary (‘dll.dll’)
```

Once the infected dll has been prepared, the \knowndlls\msi.dll section is recreated to point to an infected dll, and the msiexec.exe service is started to force the now infected library to be loaded:

```
NtOpenSection(..”\knowndlls\msi.dll”..)

NtMakeTemporaryObject(..) // clear the OBJ_PERMANENT flag from section

CloseHandle(..)

NtCreateSection(..”\knowndlls\msi.dll”, .. <temporary_file>..) //recreate the msi.dll section, now pointing to the infected msi.dll library in <temp filename>

..

StartService (..”Windows Installer”..)
```

The main idea of this technique is that, since it is executed in the context of the *Windows Installer*, the malicious code will have all the necessary privileges to download and install anything. It downloads and installs a fresh build of the TDSS kernel component.

Another advantage of the technique is that no obviously malicious behaviour is exhibited, so a HIPS will fail here until it 'learns' this particular trick.

The dll.dll functionality itself is quite simple, as can be seen in the flowchart shown in Figure 1.

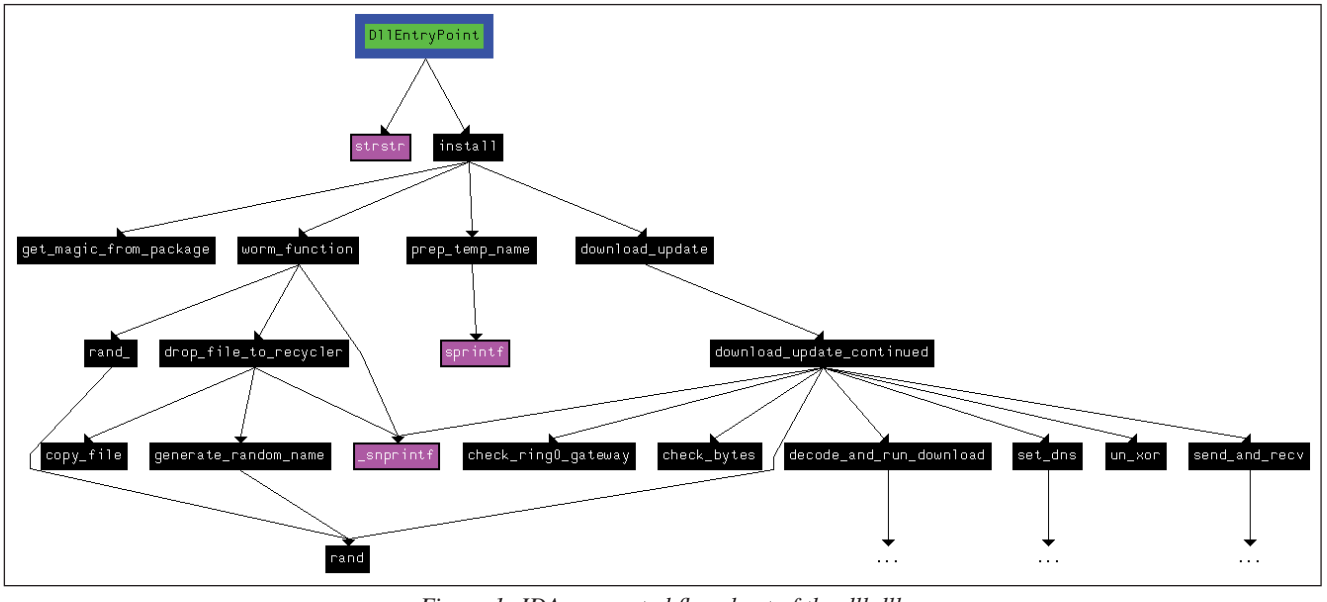


Figure 1: IDA-generated flowchart of the dll.dll.

The driver

TDSS does not have its own userland executable file. All core functions are provided by a driver, which is loaded automatically at startup. High-level functions are provided by additional DLL module(s) injected into processes.

Core functions provided by the driver include:

- Hiding the trojan
- Providing a gateway into the kernel
- Distributing spoofed DNS servers to network services
- Blocking anti-virus solutions (listed in a configuration key) from loading
- Injecting a DLL into browser executables
- Installing new DLL modules.

ROOTKIT FUNCTIONALITY

The trojan hooks the following functions in the kernel:

IofCallDriver

IofCompleteReq

NtFlushInstructionCache

NtQueryValueKey

NtEnumerateKey

The latter three hooks are implemented via SDT modification. The NtEnumerateKey hook is used to hide all the TDSS registry keys listed in the trojan's configuration key ('gaopdx*' in this case), except for trusted processes.

The NtQueryValueKey hook is used to spoof DNS addresses without modifying the registry (and therefore without triggering a HIPS registry alert), via a substitution of 'DhcpNameServer' and 'NameServer' [14] registry values.

Hooks to IofCallDriver and IofCompleteRequest are implemented by splicing the kernel code in ntkrnlpa.exe in memory. They are used to hide the trojan's files and probably its network TCP activity.

A hook to IofCallDriver is used to infiltrate all the IRPs system wide, which allows the trojan to hide its own files (beginning with the string 'gaopdx*' in this case) when it catches an IRP to a file system driver:

```
If ( FsRtlIsNameInExpression (..”*\\gaopdx*” or “*\\
TEMP\\gaopdx*”..) )

Then return (STATUS_TOO_MANY_SECRETS)
```

IofCompleteRequest has a similar functionality.

Ring0 communication gateway

The NtFlushInstructionCache hook is slightly more interesting, providing a non-typical communication gateway to the driver. To make use of the gateway, one should call the NtFlushInstructionCache API as follows:

```
push  0 ; argument to the command

push  'VERG'  ; 4-byte command, allowing to prove the
hook is in place

push  'TDL2' ; a magic value which leads execution to
the command processor and not to the original API

call  ds:ZwFlushInstructionCache ; this is a piece
of code from the dll.dll component, checking for the
presence of the core driver.
```

The scope of available commands is very limited and, in contrast to some security drivers, will not allow control to be taken of the driver. Available commands include passing trojan-related variables from the kernel to userland, inserting a termination job (via a kernel APC) into a given process or thread, and maintaining installation of new DLL modules.

Persistent functionality

The driver engages ExQueueWorkItem to launch a number of kernel threads. The execution flow of the work items is looped to provide periodic execution. The three work items provide periodic renaming and re-registering of the trojan's driver ('\registry\machine\system\currentcontrolset\services\gaopdxserv.sys'), disabling of a system firewall ('\registry\machine\system\currentcontrolset\services\sharedaccess\parameters\firewallpolicy\') and other functions.

Blocking security solutions

The driver installs, via PsSetLoadImageNotifyRoutine, a system-wide callback for newly loaded modules. In the hook, a check is performed as to whether the module being loaded is included in the 'disallowed' list in the trojan's configuration registry key. The driver will prevent a disallowed module from loading.

MANUAL DISINFECTION

Manual disinfection of TDSS is trivial. The following instructions are for a generic method that will completely remove any specimen of the TDSS family. This removal method is suitable for any end-user, since it is very simple and requires neither special skills nor specific tools:

1. Go to Device Manager and turn off and delete any inappropriate 'Non PnP driver' there.

   You can search for a specific name (quadraserv.sys in this case, or gaopdx*/TDSS*/clbdriver/seneka/etc .sys in the case of a typical TDSS family member), but the name is subject to change, so it is best not to rely on it.

   After this manipulation, the worm's files and registry values that were hidden become visible, and thus possible to be removed by hand.

   Note: An anti-rootkit can be used reliably to locate the trojan's core files. *GMER* or *RkU* are the best choices; *Avira Antirootkit* also copes with the task.

2. Remove the file corresponding to the device just deleted. If there is no such file, try sorting system32/ drivers and system32/ files by creation date and remove whatever looks suspicious according to its name and content. TDSS core files consist of a .sys and one or more .dlls.
3. Search throughout the registry using the malicious device and file name strings found in steps 1 and 2. Delete all the relevant keys.
4. Remove all the <drive letter>://autorun.inf and <drive letter>://RECYCLER/*.com files, if any.
5. Reboot.
6. Launch your AV, and let it clean the rest (TMP files etc.)

Note that steps 1–4 must be carried out manually, without any anti-malware, because if an anti-malware product lacks a single signature for a trojan's core file, the file will not be removed and the malware will return after reboot.

CONCLUSIONS

- The success of TDSS proves that the bypassing of protection mechanisms is a straightforward task, for which no kind of advanced invention is necessary.
- Malware writers continue to explore unobtrusive ways of bypassing protection [15]. In the case of TDSS, the skilful utilization of a whitelisted application to download and install malware is observed.
- Bundling malware together with legitimate software is an effective technique (though not new). The idea is that if a user is intentionally launching an application, s/he will probably skip any security alerts, including driver installation alerts (which are quite normal, for example, in the case of a video codec installation [5]) and UAC. Furthermore, some behavioural protection solutions might be fooled by the visible application window.
- Redirecting a whole network's DNS traffic to an attacker's service is an extremely important innovation, since it allows for the transparent delivery of malware to clean machines, as well as serving malicious redirects. It's almost like a new kind of worm functionality.

Behavioural protection/HIPS developers should consider keeping an eye on the behaviours/actions that allow TDSS to succeed:

- NtOpenSection, NtMakeTemporaryObject and other functions allowing tampering with system sections.
- Accessing a system DLL file.

- LoadLibraryEx with a parameter of DONT_RESOLVE_DLL_REFERENCES, which is used by dll.dll to load the original msi.dll.
- Tampering with system DNS and DHCP configuration.
- PsSetLoadImageNotifyRoutine. Though a protection may be turned off by the time this API call is made, it may not be.

Although most of these actions are not malicious by themselves, they clearly pose a minor threat and thus should be considered in combination, supplied with reasonable threat weights, and within a particular process execution context.

REFERENCES

[1] http://www.google.com/search?q=tdss+%7C+tidserv+%7C+tdsserv+daterange:01012009-26042009+inurl:forum.

[2] http://www.kaspersky.com/viruswatchlite?search_virus=TDSS.

[3] http://ddanchev.blogspot.com/2009/01/embassy-of-india-in-spain-serving.html.

[4] http://mad.internetpol.fr/archives/3-Etude-de-cas-Infection-rootkit-TDSS.html.

[5] http://www.threatexpert.com/report.aspx?md5=2c5c874235a73fc50a69780c7ad1488a.

[6] http://www.threatexpert.com/report.aspx?md5=d2ada2dba8e036d37726ebddbcc9e9d6.

[7] http://www.threatexpert.com/report.aspx?md5=b17d76537ef5d94547fc4ca8851b35da.

[8] http://www.symantec.com/security_response/writeup.jsp?docid=2008-091809-0911-99.

[9] http://www.infobyte.com.ar/down/isr-evilgrade-Readme.txt.

[10] http://www.symantec.com/security_response/writeup.jsp?docid=2009-032211-2952-99.

[11] http://www.anti-malware.ru/node/1250.

[12] http://www.f-secure.com/v-descs/backdoor_w32_tdss.shtml.

[13] http://www.virustotal.com/analisis/122e4ade1c0fa88cbab02880a3b2ed98.

[14] http://technet.microsoft.com/en-us/library/cc962470.aspx.

[15] Shevchenko, A. Advancing malware techniques 2008. Virus Bulletin, January 2009. http://www.virusbtn.com/pdf/magazine/2009/200901.pdf.

PRODUCT REVIEW

COMODO INTERNET SECURITY

John Hawes

Comodo has long been something of a mystery to me. Best known for its very highly regarded firewall products and more high-level security solutions, the company has also built up a considerable reputation and following for its anti-virus offering, but has yet to take part in *VB*'s certification tests and is not well represented in other independent tests either. Among the small group of anti-virus products never to have joined in a VB100 comparative review, *Comodo* has by far the most vocal supporters. We receive more queries about the product's non-appearance in our tests, and requests for information on its performance, than for any other product.

On receipt of the first of the many inquiries about the firm a number of years ago, I checked out its various websites and made contact, discovering that the company's product was being made available free of charge as a long-term beta project – which was considered unsuitable for testing at that stage. The beta phase went on for quite some time, with a string of ancillary products emerging alongside the initial anti-virus, and continued to build up reputation and interest. With a full product range properly released several versions ago and now well established, it seemed high time for the *VB* test team to take a quick look at the company's current flagship product, the *Internet Security* suite.

WEB PRESENCE AND SUPPORT

Comodo's online presence has a very slick and professional look and feel, with the company's wide range of security-related tools and solutions mostly presented in separate microsites, making the main www.comodo.com perhaps a little more product-oriented than the sites of many security vendors. The range of desktop security solutions is presented in the main body of the front page, but the pull-down 'Products' list focuses almost entirely on a wider selection of business-oriented security solutions – secure messaging and web access, code and website signing, VPN, backup and compliance solutions, and much else besides.

While the standard likes of company news and general security information are present, it is the product range that takes centre stage, with the company's marketing strategy for its desktop anti-malware and firewall products immediately obvious. The strategy of making basic versions of the products freely available for home use, while charging for the more multi-layered, well-supported 'pro' and corporate versions, is one which has proved highly successful for many vendors. Here, just about every product in the range is offered as a free, standalone edition, and even a suite is made available without charge. This strategy is not simply a marketing tool, however – as anti-malware increasingly makes use of distributed global knowledgebases, with users contributing information on safe applications and behaviours, strong market penetration and a broad user base have become increasingly important aspects of the protection provided by a product. Many users will be baffled by popup alerts that provide highly technical information about an application or activity and which require the user to make some decision as to what course of action should be taken. This problem is mitigated by the provision of 'herd' information on the alert: the opinions of the collective are provided as an aid to decision making. The issue of whether the opinions of the collective can be trusted remains a thorny one, but at least such systems provide some assistance.

The user community performs another function in the form of the support provided by online forums. *Comodo* hosts some bustling and well-administered forums and FAQs, providing a wealth of information and assistance on the full range of products, and the company's users – expert and otherwise – are similarly well represented on several other popular security forums.

For those choosing to pay for a more advanced product, the level of support offered is one of the most important decision-making factors – and here lies one of the most distinctive selling points of *Comodo*'s flagship line. A complete support package is available as an upgrade to the standard subscription, with the support provided directly to the user's PC via a proprietary remote access system. This allows the firm's techs to get in and fix issues with their customers' systems without the need for complex and difficult explanations to inexpert users over the phone or email. Indeed, the copy of the product we were provided with for review came with the offer to have it remotely installed and configured by an expert. The support offering extends far beyond the basics of setting up the product and dealing with the problems caused by it or any malware it fails to detect – it also seems to cover just about any PC-related issue the customer may have, from installing software to setting up printers. This is not something I am aware of many other vendors providing, and it makes for a pretty impressive unique selling point for *Comodo*. Unfortunately there was insufficient time to test the service with any complex issues, and since much of our interest lay in checking out the performance of the anti-malware engine, we opted to do our own installations in the *VB* lab.

INSTALLATION AND CONFIGURATION

The set-up process runs along fairly standard lines. The first item of note is that the product is included as a complete download, rather than one of the tiny download-and-install

systems that seem to be growing in popularity with vendors these days, and appears to be updated fairly regularly. This pleased me, as working in the security industry and thus being somewhat paranoid, I always like to have security software installed, running and reasonably up to date on any new system before I think about connecting it to the Internet– which is not always possible with some solutions.

The initial stage of the installer presents a rather clunky self-extracting dialog, which hovers in the background throughout the install process, but the installation GUI itself is much more slick and attractive. It runs through the standard stages of EULA, choice of install location and space requirements (the product needs a minimum of 123 MB of hard drive space – not too much of a strain for any modern system), and then some further options on which components to install – the suite can be used as just the firewall, just the AV, or both (which is the default setting). Next comes the option to contribute to the 'Threatcast' community collaboration system, with ample information provided on how this works and what kind of data might be shared. A final component is offered in the form of a browser toolbar in collaboration with the *Ask* search engine. This offers to reset the default browser search to *Ask* and the homepage to *Comodo*. Both of these are active by default, which I'm not too keen on, but this seems to be pretty standard with toolbars. After a few further steps of finalizing, connecting to the community system and activation, a reboot is required to finish things off.

The main interface of the product is pretty impressive: it is quite attractive and well laid-out, uncluttered and clear. Status information is provided on various aspects of the product, the anti-malware and firewall systems, in simple terms with easy links to run scans or lock down the firewall. Further data on active processes and connections is also included.

More information and options for the various components are accessed via separate pages for the anti-malware, firewall and 'Defense+' HIPS systems, with a 'miscellaneous' tab providing the likes of interface password and language options, updating, suspect file submission, access to online forums, and help. Each of the main areas provides a good range of controls for the given module, and each is accompanied by clear and simple explanations of the options available. Of these, the anti-malware is likely to be the most straightforward for the majority of users, with the controls for the firewall and HIPS systems likely to need a little more effort for the average inexpert user to comprehend, while some of the options provided in the advanced areas are likely to be unsuitable for any but the keenest users. Help is generally available however, and with a little application and research all of these options can be used to improve and enhance the level of security provided.

As with all security products, the 'set and forget' approach is only as good as the default settings. Here they seem pretty sensible across the board, but to get the most out of any product the user needs to invest some time to study and understand the threats they face and how they can best be mitigated – something we encourage all users to do.

MALWARE DETECTION AND SYSTEM PROTECTION

Having familiarized ourselves with the layout of the product we got down to our principal area of interest: the detection capabilities of the anti-malware engine. Having confirmed that the on-access component was fully operational and having disabled the warning popups (which would have seriously impeded on-access tests), we ran the product through most of the standard VB100 tests using the same systems and test sets as used in the most recent comparative (see *VB*, April 2009, p.15). As the product had not been frozen on the correct deadline for that test, this would not provide results that could be compared scientifically against those of the large field of entrants last month, but we hoped it would at least provide a general overview of the product's abilities.

The first hurdle here was determining the exact date of the product as downloaded. On installation an attempt was made to update, and indeed the update status claimed that the product had been updated on the day of the install. As we were running the product on a machine in a sealed-off part of the *VB* test lab, with no access to the Internet, this was somewhat baffling. To get around this, we also took the updates from an Internet-connected system and used them in a second run through the tests, this time using updates confirmed to be almost exactly a month more recent than those required for the original test that used these sets.

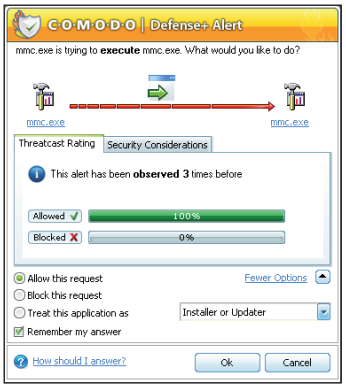
The first things we looked at were scanning speed and false positive rates, running the product through our full standard clean sets. This proved very impressive, with a few

suspicious alerts on unusual packers but no full false positives at all – a remarkable achievement given the problems our test sets have been known to cause products in the past, and given that this was the first time the sets had been checked with this scanner. Scanning speeds were pretty good too, perhaps not quite at the very top of the field but comparing favourably with most of the products included in the last comparative test. This proved true in both on-access and on-demand modes, demonstrating on-access overheads that were well within acceptable levels. Indeed, at no point throughout the testing did we observe any untoward slowdown on our systems, even when running under heavy strain.

Moving on to the malware detection side of things, we ran through the complete set of test samples used in the last VB100, and here things were perhaps not quite so impressive. In some sets detection rates were fairly high, coming in at around 95% on the older set of worms and bots and also on the WildList set (although here we would expect nothing less than 100% in a top-quality product). The trojans set, containing samples first seen a few months prior to the test, was reasonably well covered at a level on a par with much of the mid-field in the last comparative, but the more recent RAP sets were not so well handled. It was in the polymorphic sets that the most worrying performance was seen however, where there was very little coverage at all. With the test sets including numerous variants of the nasty and complex W32/Virut family, none of which were detected, this is clearly an area that should be improved.

Another issue that quickly became clear was one of instability. Several times during our attempts to get through the tests the product experienced problems, crashing on a fairly regular basis. This seemed to occur only during on-demand scanning, and while the on-demand scanner generally refused to initiate any further actions until a reboot, in most situations it seemed that the on-access protection remained active. Only on one occasion did the product become fully nonoperational. All of these problems occurred during intensive scanning of large numbers of infected samples. They appeared not to be directly related to any specific sample, as on subsequent occasions the same sets were scanned without difficulty. The pattern of crashes hinted at there being some issues related to the handling of large numbers of detections in a short period, perhaps not helped by the product's impressive speedy scanning rate over infected items. The problems thus seem unlikely to affect the real-world user, but nevertheless we will provide full details to the developers to ensure nothing more serious underlies our experiences.

Not all is doom and gloom however, as the basic static detection is not the only protection feature available. The HIPS system, dubbed 'Defense+', combined with the outbound portion of the firewall, offers an extra layer of defence, and trying this out against some of the samples that the on-access anti-malware component had allowed to run provided much more encouraging results.



Running numerous items against the product's filters and hooks, it seemed that nothing we could throw at it would be allowed to operate completely uninhibited. Most of the more serious malicious activities, such as doctoring or creating registry entries, dropping files in system folders or drive roots, initiating network connections, injecting code into memory or running processes and so on, were blocked or at least alerted on. While popup alerts are not always the most useful tool, with many users likely to grow frustrated by them and simply click 'OK' regardless of the message text, they do at least provide some protection against malicious activities, and hopefully users are growing more alert to the dangers of malware. The popups are also supported by the opinions of the community system which, in most cases, seemed to advise taking the most sensible course of action.

Of course, not every activity of the malware was entirely prevented. While most clearly unwanted behaviours were easily brushed aside, files were not stopped from being dropped into unprotected areas, and some apparently less significant tweaks were allowed to be made to the registry and other settings. This is less than ideal, and users may want to run occasional checks with additional software to clean up any potentially dangerous remnants – which is a good policy with any anti-malware solution. The product range also has a strong reputation for post-infection cleanup, which unfortunately we did not have time to investigate in any depth; we hope to develop additional metrics to measure such things in the near future.

OTHER FUNCTIONALITY

The firewall is one of *Comodo*'s main strengths – it is one of the most highly regarded on the market, and from a quick

run through seems to perform excellently. The basic set-up is fairly rigorous, and configuration is available both at a basic level and in depth. Simple sliders provide various levels of paranoia, from fairly lax and trusting to complete lockdown, and the default provides a happy medium with not too many alerts and popups. The advanced configuration options provide a wealth of fine-tuning, presented with clarity and simplicity, but as is generally the case with such things, a minimum level of understanding is required to ensure the right changes are implemented properly. A few options proved rather difficult to locate, but generally the layout made good sense and after some familiarization nothing we could have wished for was lacking.

The same is true of the HIPS system, which provides a similarly intensive level of configuration in a pleasingly similar style, making for good continuity across the two modules. The addition and fine-tuning of filters focusing on particular areas of the system or registry, particular file types, specific applications and even developers is laid out in a highly usable manner, in the same way that the firewall provides fine-tuning of filters of network zones, connections, ports, and applications. While some members of the test team would have liked to have seen some additional areas monitored by default, most of the standard settings seemed pretty thorough, and the level of control easy to adjust via another paranoia slider.

Another component of the suite is the optional toolbar which, along with some standard items from *Ask*, includes *Comodo*'s 'SafeSurf' technology, designed to monitor memory for buffer overflow attacks and similar web-based threats. Not being big fans of toolbars in general, and being rather short of time, we didn't investigate this thoroughly, but it seemed to offer some useful protection (although we noted that some of the associated data-gathering and other toolbar tactics have come in for some criticism from various online commentators).

One final module which deserves a mention is a process viewer, which displays all running processes along with some brief details and provides the option to terminate anything that is unwanted. This is another tool that will be of most use to the well-initiated, but again it is presented in a clear and simple style with good usability.

CONCLUSIONS

Having approached *Comodo*'s product as an almost completely unknown entity, the overall impression it has left after an all-too-brief acquaintance is a favourable one. The design is both visually appealing and easy to navigate – which is not always an easy combination to pull off. The multi-layered protection seems to provide a pretty decent standard of security with the default settings and offers a really quite excellent depth of configuration. The user community backing it all up is clearly highly active and committed, which are vital components in any herd-based system. The additional support offering, covering a vast range of computer support needs, is just about unique.

Although we encountered a few issues with the anti-malware scanners, including some less than excellent detection rates, these should improve as the company becomes more established, gets more involved in testing and improves relations with the rest of the industry. The stability issues we encountered in our intensive tests were also fairly minor, and unlikely to affect most real-world users. The only other downside to the product is a fairly large number of popups, particularly during the initial stages of use. Although these will generally be assisted by the group consensus data, they do require some decision-making from the user. They can also be tweaked and configured to be more automated and less intrusive, but again, the users will have to apply themselves to ensure the appropriate settings for their situation.

As we have commented many times before, to get the most out of any security product, users have to make some effort to learn how their computers work and what effect their decisions will have. Perhaps those who are not willing to do so should not expect to operate with complete impunity in an online world riddled with criminals and con men; for those who understand both the threats and how to defend against them, this product provides the full range of control necessary to provide a highly secure environment.

Technical details

Comodo Internet Security 3.8 was variously tested on:

AMD Athlon64 3800+ dual core, 1 GB RAM, running *Microsoft Windows XP Professional SP3* and *Windows Vista x64 SP1*.

Intel Atom 1.6 GHz netbook, 256 MB RAM, running *Microsoft Windows XP Professional SP3*.

END NOTES & NEWS

The 18th EICAR conference will be held 11–12 May 2009 in Berlin, Germany, with the theme 'Computer virology challenges of the forthcoming years: from AV evaluation to new threat management'. For more information including programme details see http://eicar.org/conference/.

SEaCURE.IT will be held 19–22 May 2009 in Villasimius, Italy. SEaCURE.IT, the first international technical conference to be held in Italy on security-related topics, is aimed at bringing together leading experts to create a unique setting for networking and discussion among the speakers and the attendees. For details see http://www.seacure.it/.

NISC 10 will take place 20–22 May 2009 in St Andrews, Scotland. For more details including provisional agenda and online registration see http://www.nisc.org.uk/.

SecureScandinavia takes place on 2 June 2009 in Stockholm, Sweden. The one-day conference will focus on emerging threats, discussing the importance of digital control systems and the protection of and guidelines for SCADA systems. Presentations will focus on the practical implementation of security measures for critical infrastructures. See http://www.isc2.org/EventDetails.aspx?id=3810.

The 21st annual FIRST conference will be held 28 June to 3 July 2009 in Kyoto, Japan. The conference focuses on issues relevant to incident response and security teams. For more details see http://conference.first.org/.

A Mastering Computer Forensics masterclass will take place 22–23 July 2009 in Jakarta, Indonesia. This intensive hands-on training course covers topics including: the tools required to perform a number of basic forensics techniques; methods and procedures to maximize effectiveness of evidence gathering; and legal and process issues surrounding incident response, litigation support and preserving evidence for presentation in a court of law. For details see http:// www.machtvantage.com/.

Black Hat USA 2009 will take place 25–30 July 2009 in Las Vegas, NV, USA. Training will take place 25–28 July, with the briefings on 29 and 30 July. Online registration is now open and a call for papers has been issued, with a deadline for submissions of 1 May. For details see http://www.blackhat.com/.

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

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

VB2009 will take place 23–25 September 2009 in Geneva, Switzerland. Early bird registration rates apply until 15 June 2009. For the full conference programme including abstracts for all papers and online registration, see http://www.virusbtn.com/conference/vb2009/.

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

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

RSA Europe will take place 20–22 October 2009 in London, UK. The 2009 conference celebrates the life and work of writer and poet Edgar Allan Poe and his influence on the field of cryptography. Online registration opens 12 May 2009. For full details see http://www.rsaconference.com/2009/europe/.

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/

vb Spam supplement

CONTENTS

NEWS & EVENTS

INDICTMENT FOR UNIVERSITY SPAMMERS

A group of four spammers were indicted by a federal grand jury in Missouri last month after they masterminded an extensive spamming campaign targeting more than 2,000 colleges and universities across the US.

It is believed that the group developed email-harvesting programs that were used to obtain more than eight million student email addresses. The campaign began at the University of Missouri, where one of the four was studying at the time, then spread, eventually targeting almost every college and university in the country.

In total, the four face more than 50 charges including fraud in connection with computers, fraud in connection with email, conspiracy, and violations of the CAN-SPAM Act. The defendants face up to ten years imprisonment and the indictment also contains a forfeiture allegation which, upon conviction, would require them to give up their $4.1 million in proceeds.

EVENTS

The Counter-eCrime Operations Summit will be held 12–14 May 2009 in Barcelona. See http://www.antiphishing.org/.

The 16th general meeting of the Messaging Anti-Abuse Working Group (MAAWG) will be held in Amsterdam, The Netherlands, 9–11 June 2009. See http://www.maawg.org/.

Inbox/Outbox 2009 takes place 16–17 June 2009 in London, UK. See http://www.inbox-outbox.com/.

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

FEATURE

MAIL AUTHENTICATION WITH DOMAIN KEYS IDENTIFIED MAIL – PART TWO

John Levine
Taughannock Networks, USA

In last month's article (see *VB*, April 2009, p.S1), we learned about the mechanics of DKIM, DomainKeys Identified Mail, a message authentication system that has recently been standardized by the IETF. DKIM allows a signer to add a DKIM-Signature header to a mail message. The header includes a hash of the message body and headers, and a cryptographic signature that can only be decoded by a key stored in the signing domain's DNS. The recipient(s) of the message can check the signature to verify both that the message has not been modified since it was signed and, using the decoding key from the DNS, that it is a genuine signature from the signing domain.

HOW DKIM FITS INTO MAIL FILTERING

It is surprisingly tricky to integrate DKIM into mail filtering setups, not because the authentication system is inherently hard to use, but because most current filtering technology is based on weeding out unwanted mail, and DKIM doesn't fit into that model. DKIM will be most useful for recognizing mail from known good senders who sign their mail. This can be achieved by whitelisting mail with those senders' signatures.

The key to effective use of DKIM is for each message to have a signature that the recipient recognizes. If a message has a signature with a d= domain that is included on a recipient's whitelist, the recipient's mail server can safely skip the spam filters and just deliver it. This works even when the d= domain doesn't match the From: line header. For example, on my system, my users have about a dozen personal domains that they use in their outgoing mail, but I know them all and am confident that they will behave themselves, so I put a signature on all of the outgoing mail with my own domain. This lets them all benefit from a receiver's single whitelist entry.

Large mail systems already maintain reputation databases that track the sources of good and bad mail. When an *AOL*

user clicks the spam button to complain about a message, *AOL* updates its reputation database about the source of the message. Currently message sources are tracked by IP address, since that's the only reliable source identity available, but as DKIM signatures become more common, they will become the preferred identity.

Domains are much better identity handles than IP addresses. They are more stable and don't change if a sender switches ISPs, or if a change in circumstances requires new mail hosts with new IP addresses. They also provide a more useful granularity in situations where a group of senders share a small set of IP addresses, such as at a shared web host or email service provider. When a domain has been in use for a while and is widely recognized as having a good reputation, the domain's value will increase and its owner will be encouraged to be careful with its mail to preserve that value.

Which signatures to use

A message may arrive with multiple signatures, some of which validate and some of which don't. Which signatures should a receiver use? DKIM has not been around long enough to provide an answer based on experience, but if using it for whitelisting, the logical answer is to use the valid signature with the best reputation.

There are both innocent and malicious reasons as to why a message might have invalid signatures. Intermediate mail hosts may mutate a message in ways that break a signature, such as tidying up header lines, or it may have been sent through a mailing list (a topic addressed later).

Creating a correct DKIM signature involves some tricky programming, so a new signer may just have buggy code that sometimes generates broken signatures. A test event in late 2008, attended by many DKIM developers including several of those who wrote RFC 4871 [1], found a lot of obscure compatibility issues and generated 15 separate RFC errata clarifying parts of the spec [2].

Just as bad guys add fake Received: headers to spam to try to disguise the true source of their messages, they will also likely add fake DKIM signatures. Trying to guess the difference between innocent broken signatures and forged signatures would be just as hard as any other kind of heuristic spam detection, so recipients just ignore broken signatures.

Having limited our attention to valid signatures, the other part of the question is what to do if there are more than one. Imagine a message that has two signatures, the first of which is from someone you trust and the second of which is from someone you don't. You can't tell whether they were both applied at the same time, or whether they were applied sequentially as the message passed through different mail systems. But if you really trust the first signer only to sign mail you want to receive, why does it matter if the message passed through a bad neighbourhood on its way to you? Since the signature validated, you know that the message you got was the same one they signed, so the message should be good.

VOUCH BY REFERENCE

While large ISPs can afford to maintain their own whitelists and reputation databases, doing so is beyond the ability of small operators. For many years, receivers have used shared DNS blacklists of IP addresses that send unwanted mail – examples include the *Spamhaus* SBL and XBL. Shared blacklists of bad domains are unlikely to be useful, however, since the supply of domains is unlimited and bad guys will just discard any that appear on blacklists. On the other hand, good domains are quite stable, so shared whitelists of good domains will be of great use.

A small consortium called the Domain Assurance Council, (of which I was one of the directors) has designed a shared whitelisting system called Vouch by Reference (VBR) [3], which is scheduled to be issued as an IETF standards track RFC. VBR can be used with other authentication schemes such as Sender-ID, but we designed it to work with DKIM.

VBR provides a very simple way to publish a list of domains about which the publisher wants to make a positive statement. There is no standard term for what VBR does, but it is most often called certification. If a sender expects a certifier to vouch for its mail, it puts a VBR-Info: header into each message:

```
VBR-Info: md=bigbank.com; mc=transaction;
mv=certifier.com:certifier-b.com;
```

The md= domain must match the d= domain on a valid DKIM signature on the message. The mc= field is a message category asserted by the sender. This can be 'transaction', 'list', or 'all', to say that the message is related to a transaction, that it has been sent to a mailing list, or that it is some other kind of mail. The mv= field is a list of domains of certifiers that the sender expects to vouch for them.

VBR publishers have a set of VBR records, one for each domain they certify. Each VBR record is just a DNS TXT record whose name is the certified domain, the token _vouch, and the voucher's domain:

```
bigbank.com._vouch.certifier.com TXT "transaction"
```

The content of the record is a space-separated list of words from the set 'transaction', 'list', or 'all', to indicate that the publisher vouches for transactional mail, list mail, or all mail from that domain.

To check whether the sender of a message is certified, a receiver first ensures that there is a valid DKIM signature with an appropriate domain. Then it checks to see if any of the certifiers in the mv= list are certifiers that it trusts. (The mv= list is an optimization to avoid having to check certifiers that aren't likely to have vouching data. Since bad guys can set up their own certifiers, receivers should only check certifiers they know.) Assuming there's a good signature and at least one known certifier, then the receiver looks up the VBR record, and if it exists and its content agrees with the mc= category, the certifier has vouched for the sender. This sounds complex, but it is quite a fast process since it involves at most a single DNS lookup per publisher.

Some VBR publishers might want to assert that all the mail from the domains in its list will be worthy of delivery, but I expect VBR to be more useful to identify groups of organizations of a particular type. For example, the FDIC (the agency that insures banks in the United States) might publish a list of the domains of its member banks and vouch for their transactional mail. The FDIC can't promise that its banks won't send you unwanted ads, since that's legal in the US, but they can at least assert that a signed message from a domain in their list is really from the bank and isn't a phish.

Each receiving system can choose which VBR publisher to use to whitelist signed mail by domain, just as it chooses now which blacklists it trusts to block mail by IP. The current version of VBR effectively communicates, one bit per lookup, that a domain's mail is good. We considered more sophisticated VBR data such as reputation scores, but decided that at this point we don't understand reputation systems well enough to design a scoring system that would be broadly useful.

DKIM AND MAILING LISTS

One of the most confusing application areas for DKIM is mailing lists. Some are 'announcement' lists, where all the messages are sent by a single party, while others are 'discussion' lists, where members can send in messages which are then passed on to the entire list. Each presents its own challenges.

Announcement lists

Announcement lists, particularly those used for advertising, are often outsourced to specialist companies known as Email Service Providers (ESPs) which handle the mechanics of list management and delivery issues. Depending on the ESP, the mail may appear to come directly from the ESP's client, with the involvement of the ESP visible only by looking at mail headers, or the ESP may co-brand the mail with the client. Large clients tend to do the former, small clients the latter.

In each case the ESP (usually with the client's advice) needs to decide what signatures to put on each message, and for signatures in the client's domain, how to manage the signature keys. For the relatively invisible ESPs, the signature is typically the client's, which means that the DKIM validation key has to be installed in the DNS under the client's domain. One way to do this would be for the ESP to generate the key records and give them to the client to install, but that doesn't work well, since clients' DNS management skills vary widely, to put it politely. A more workable approach is for the client to delegate part of their DNS tree to the ESP.

As a real example, online travel agent *Orbitz* uses ESP *Responsys* to manage its weekly online newsletter. The company has delegated the subdomain my.orbitz.com to *Responsys*' name servers. The newsletters have a return address of orbitz@my.orbitz.com, and the DKIM signature d= domain is also my.orbitz.com. This allows *Responsys* to handle all of the DKIM mechanics, while maintaining orbitz.com as the responsible party. In particular, if *Orbitz* were to switch ESPs, the company would take the reputation of my.orbitz.com with it, since it ultimately controls its delegation.

At the other end of the spectrum, *Constant Contact* is an ESP that provides a service to tens of thousands of mostly tiny businesses with small lists and small mailings. In their case, it makes sense to sign mail with both the client's domain and constantcontact.com, since many individual clients will have too small a mail volume to get much of a reputation, while the ESP's aggregate volume is large enough and its list management is good enough that many receivers would be willing to whitelist mail that it has signed. (I don't believe it has worked out the mechanics of its client signing yet.)

Discussion lists

Discussion lists present a different set of identity issues, since each message sent through such a list has a From: address of the original contributor, even though the list sent it to the list members. This has engendered a great deal of confusion among DKIM implementers. Some lists make few enough changes to the messages they pass through that a DKIM signature on incoming messages might still be valid when received by list members. DKIM includes a few features for list mail, such as an optional message length field which is intended to let recipients skip the footers that are added by list software. One theory says that list

software should refrain from making any changes that will break signatures, so recipients can apply their reputation and filtering rules based on the original senders. This seriously misunderstands the way that DKIM works (in my opinion at least), and is unworkable with modern list software anyway.

It is a rare list package that doesn't break the signatures on its messages. Something as simple as adding the list name to the Subject line will do so, and modern list software often rewrites list bodies, deleting attachments, turning HTML into plain text and vice versa, and in some cases such as *Yahoo Groups*, rewriting the HTML of the message to add a message footer. Fortunately, there's no need to preserve the signatures on the messages, because for the recipients, the signature and reputation that matters is that of the list, not of the individual contributors.

When someone subscribes to a list, they do so (presumably) because of the list's contents, and they depend on the list's operator to control what mail the list sends. It is perfectly reasonable for the list management software to perform DKIM checks on its incoming mail as part of the process of deciding what messages to accept, but once a message is accepted, the list software puts its own signature on its outgoing mail, and that's what the recipients use. Advocates of preserving incoming signatures ask 'what if bad guys send forged mail to lists?', to which the reasonable answer is that list managers will deal with it, just as they've dealt with other kinds of abuse over the past 40 years.

At the moment, most of this argument remains hypothetical since relatively few lists do anything with DKIM at all, but we are starting to see lists sign their outgoing mail with a list signature, which should encourage recipients to use that signature in their mail management.

ADSP AND PHISHING

Some domains are subject to heavy phishing attacks, some of the most notable examples being *PayPal* and *eBay*, and online greeting card sites like *Blue Mountain* and *American Greetings*. In the former cases the phish is trying to steal credentials, in the latter it is trying to trick users into clicking a link that will install malware on their PCs. If one knew that all of a domain's legitimate mail were signed, it would be possible to reject some phishes by rejecting mail purporting to be from that domain but without a signature.

ADSP, Author Domain Signing Practices, is an add-on to DKIM that allows a domain to publish its practices and state that it signs all mail that includes its domain on the From: line ('all'), or that it signs all of its mail and it considers itself to be a phishing target so it wants you to throw away unsigned mail ('discardable'). ADSP is currently in the midst of design arguments. These are partly about its basic utility, since there's little reason to believe that the domains that would publish 'discardable' ADSP would all be or even mostly be actual phish targets. The other arguments are over whether the i= field should be used in the signature to force it to match the entire From: address rather than just the domain.

Whether or not ADSP is published, there are a few heavily phished domains that really do sign all of their mail, paypal.com being the prime example. Recent versions of the popular *SpamAssassin* filtering package have an ADSP option that uses a short built-in list of phishing targets to mark unsigned mail as spam.

Discarding unsigned mail from phishing targets is unlikely to make much practical difference, since it's easy to send phishes without using the target's own return address. For example, with a From: line like the one below, many popular mail programs will display the PayPal address in the comment, rather than the actual rotten.biz return address that could be signed with an ADSP-compatible signature:

```
From: security@paypal.com <evil@rotten.biz>
```

Effective measures against phishing will depend on highlighting the good mail, perhaps with an enhanced VBR that shows a brand logo (e.g. 'look for the golden dollar sign' on mail from an FDIC member bank) so people come to understand that if it's not highlighted, it's not really from a bank, or from *eBay*, or from the greeting card company. DKIM can authenticate the real mail, but it's just one part of a total package.

SUMMARY

DKIM is an authentication system that provides an effective way to assign a stable identity to mail messages beyond ad-hoc identities based on IP addresses and message From: addresses. It shows signs of wide adoption, already being used by *Yahoo*, *Google*'s *Gmail*, and many email service providers. In combination with whitelists, certification and reputation systems, it will be a key tool to separate mail that recipients want from mail they don't want.

REFERENCES

[1] http://tools.ietf.org/html/rfc4871.

[2] http://www.rfc-editor.org/errata_search.php?rfc=4871.

[3] http://www.domain-assurance.org/protocol-overview.phtml.

COMPARATIVE REVIEW

ANTI-SPAM COMPARATIVE REVIEW MAY 2009

Martijn Grooten

If you happened to pass the *Virus Bulletin* office during the last few days of April, you would have been forgiven for thinking you had heard the popping of champagne corks in celebration of the completion of our first comparative anti-spam test. After months of consideration, internal and external discussion, trials and retrials, we are very pleased to be able to reveal the results of the first test.

Still, much as we believe that our test is a good one, we are the first to admit that there is room for improvement – indeed we are already working on a number of adjustments to the test set-up. Moreover, there were a couple of minor bugs that had to be fixed during the course of the test, and it is only fair that we confess to these issues.

One of the things that went wrong was that, one week into the test, the primary DNS server failed. Most products use a secondary DNS server as a backup solution, as do our own servers, and it was for this reason that we did not notice the problem until later on. The problem was brought to light when one of the products on test showed a significant drop in performance – it turned out that the product in question was only using the primary server for DNS lookups. While it is generally assumed to be best practice for products to use at least two DNS servers, this requirement had not been stipulated prior to the start of the test – we intend to make this a formal requirement for entrants in future tests. Of course, we have also learned that it is important to monitor the performance of the DNS servers closely.

A second bug was caused by a minor error in the script used to relay email to the products. This resulted in some of the emails being relayed incorrectly. Thankfully, a comprehensive logging system meant that we were able to identify these emails easily and, after fixing the bug, remove them from the test set.

THE TEST CORPUS

The test corpus consisted of all emails sent to the virusbtn.com domain between the afternoon of 9 April and the morning of 30 April 2009. The original idea was to let products filter all email, regardless of whether they were sent to an existing address, thus maximizing the amount of spam seen by the products. However, not all of the products could be configured in this way and as a result we decided to remove from the corpus any messages that had been sent to addresses that do not correspond to a genuine *VB* mailbox or alias.

After removing these, as well as the misrelayed messages, the test set consisted of 1,677 ham emails and 24,320 spam emails. The ham set included personal and business email, newsletters, mailing lists, genuine delivery failures and automated notifications. The nature of some of these emails (in particular automated notifications, newsletters and mailing lists) makes them very difficult to distinguish from spam. Nevertheless, they are all messages that the virusbtn.com end-users genuinely want to receive, and as such they should not be blocked by a spam filter. It should be noted, however, that the false positive (FP) rates recorded in this test may be higher than those reported in other tests using 'easier' ham corpora (containing fewer newsletters, mailing lists and so on). This is one of the reasons why the absolute numbers shown in the test results do not give a good picture in isolation; it is the relative numbers compared to those of other products that demonstrate how well a product performs.

To determine the 'golden standard' for each email, we first applied some ad hoc rules. For example, we determined that any message using a foreign alphabet was almost certainly spam. It should be noted that under the test regime, products are not allowed to make use of such ad hoc rules based on *VB*'s assumed email behaviour – and regular checks are carried out to ensure this is not the case. Secondly, if all products agreed on the classification of an email they were assumed to be correct; again, we performed regular checks to ensure that nothing was misclassified (even though the comparative nature of the test would mean that a mistake here would not disadvantage any product).

Finally, for all remaining emails, the golden standard was decided upon by the end-user – the *VB* employee to whom the email was sent (see p.2). To minimize the effect of human error, all emails reported as false positives by at least one of the products were double-checked to ensure the correct classification had been made by the end-user.

THE TEST SET-UP

A brief description of the test set-up follows below. Full details of the set-up and the thought processes behind it can be found in *VB*, January 2009 p.S1; *VB*, February 2009, p.S1 and *VB*, March 2009, p.S6.

A gateway Mail Transfer Agent (MTA) running qpsmtpd 0.40 on a *SuSE10 Linux* machine was configured to accept all email sent to the virusbtn.com domain. Upon accepting an email, the MTA stored it in a database then relayed it to all participating products in random order. The original email was unchanged with two exceptions: first, a Received header was added to reflect the fact that the email had passed through our MTA. Secondly, if the email

header lacked a Message ID, one was added using the mail.virusbtn.com domain.

All of the products participating in the test were configured to relay the filtered email to a back-end MTA. Where possible, they were configured to relay spam as well, and to mark spam using a special header. Using this header in combination with the IP address on which the product was located, the back-end MTA was able to link a filtered email with both a product and an email that was already in the database.

Two of the products, *ClamAV* and *SpamAssassin*, were not installed on a server; instead they were installed on the same machine that runs the MTA. For performance reasons, emails were not sent through these two products immediately after they were received. Instead, a script checked every 10 minutes for new messages then ran them through both filters.

BitDefender Security for Mail Servers 3.0.2

SC rate: 84.20%

FP rate: 1.49%

FP of total mail corpus: 0.096%

BitDefender is no stranger to *Virus Bulletin*, since the Romanian vendor is a regular participant in the VB100 anti-malware reviews. The company has also been active in the anti-spam business for quite some time and was one of the first to submit a product for this test.

BitDefender Security for Mail Servers comes in various flavours for different operating systems; the version we tested ran on a new *SuSE10 Linux* installation as an extension (milter) to the *Postfix* MTA. Installation of the product was straightforward and consisted of downloading an executable .rpm file and running it. The product can be configured using the command line, which no doubt will please many experienced *Linux* administrators, but those who prefer a graphical interface will also find themselves at ease with the web interface.

BitDefender's false positive rate was lower than that of any of its commercial competitors. The spam catch (SC) rate, however, left some room for improvement. The low spam catch score is partly explained by the product's use of only one DNS server – something the developers have since fixed. Indeed, during the period in which our primary DNS server was down, the product's performance dropped about six per cent. Despite this, the product's performance was more than decent and, while working on improvements to the product for the next test, its developers will be able to revel in the knowledge that they have already achieved a VBSpam Gold award.

ClamAV using Sanesecurity signatures

SC rate: 27.63%

FP rate: 0.00%

FP of total mail corpus: 0.00%

ClamAV is the biggest and best-known open source anti-malware product and is developed by a large group of volunteers from all over the world. While many anti-malware reviews suggest that *ClamAV*'s performance falls short of that of its commercial competitors, it still boasts many happy users. In particular, many of them use the product on mail servers to check incoming and outgoing email for malware. However, it can also be run as a spam filter, and as such it was submitted to the test. The scanning rules were based on signatures provided by a group of volunteers operating under the name *Sanesecurity*.

We had been warned that the spam catch rate would be far from that of dedicated anti-spam products and indeed, we found that the product blocked barely 28% of all spam. However, that does not render the product worthless. The fact that, even in our difficult ham corpus, no legitimate message was blocked incorrectly indicates that the product could act as a very good first-layer filter, working in conjunction with a number of others. Moreover, the nature of signatures is such that the product's performance might change significantly if it were to see a different email corpus (indeed, we saw great variation in its day-to-day performance), and I will be very interested to see how it performs in the next test, using a larger spam corpus.

MessageStream (Giacom)

SC rate: 96.50%

FP rate: 3.16%

FP of total mail corpus: 0.204%

Giacom's *MessageStream* is a hosted solution that takes the spam filtering away from the customer's mail server: email is passed through and filtered by *MessageStream*'s servers, where spam is quarantined and only presumed ham messages are sent back to the customer's mail server.

An attractive and intuitive web interface is available for the configuration of product settings as well as for the whitelisting of email addresses or full domains on either a global or personal level. I was charmed by the information that is provided on why emails have been marked as spam – enabling users to modify filter rules even if they aren't experts on spam filtering. I was less excited by the fact that there is no facility for an administrator to search all email

(sent to all addresses) simultaneously, but end-users' privacy is more important than saving the system administrator a few minutes' work.

On the company's website, the product is claimed to block at least 97% of spam and our test results indicate a similar score – far above the average. Unfortunately, there were a few false positives, but judging by the spam scores for the emails in question, most of them could probably have been avoided (albeit at the cost of a lower spam catch rate) by tuning down the spam filter slightly. A VBSpam Gold award is thus very well deserved.

M+Guardian (Messaging Architects)

SC rate: 94.83%

FP rate: 2.27%

FP of total mail corpus: 0.146%

For those companies who want to keep their anti-spam solutions in house, yet do not want to configure a server themselves, a hardware appliance might be the right choice. One of the many available on the market is *M+Guardian*, a product from Canadian company *Messaging Architects*. The appliance can be stored in a server room like any other server, with the difference that you don't have to worry about installing and maintaining an operating system.

Like most products, *M+Guardian* comes with an easy-to-use web interface for product configuration and the monitoring of email flow. I liked the fact that there was an option to send a warning once the number of spam emails received by a single user has exceeded a certain threshold – thus reminding end-users that using one's address sensibly is the first step to minimizing spam.

While the product did generate some false positives, the number was lower than average. Add to that a very high spam catch rate and *M+Guardian*'s developers can be proud to be the first to achieve a VBSpam Platinum award.

SpamAssassin

SC rate: 61.41%

FP rate: 1.07%

FP of total mail corpus: 0.069%

With a history dating back to 1997, *SpamAssassin* is the Methuselah among anti-spam products. The product is far from retirement though, and it is used as heavily as ever and still worked on by a large group of volunteers. Operating under an *Apache License 2.0*, the product is free and open source. For this test, we used version 3.1.8 on *SuSE10 Linux*, which was updated every hour.

I do not believe that using free anti-spam software is necessarily a better idea than using a proprietary product, nor do I think that the performance of a free product is bound to be worse than that of a commercial product. However, the vendors of commercial products need good reason to expect customers to pay for their wares if decent free alternatives are available – so it will be interesting to see how performances compare.

Unfortunately, *SpamAssassin*'s spam catch rate was a disappointingly low 61% – which was barely compensated for by a very low false positive rate. Undoubtedly *SpamAssassin*'s developers will be as curious as I am as to whether the low spam catch rate was caused by loose filter rules that need tightening, or whether other factors have also played a role.

Webroot E-Mail Security SaaS

SC rate: 97.57%

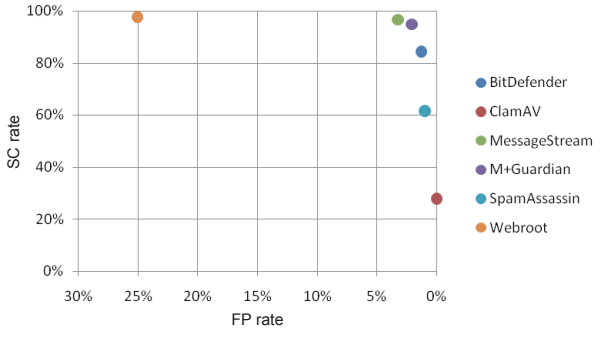
FP rate: 26.12%

FP of total mail corpus: 1.685%

Webroot is another vendor that will be familiar to *VB* readers from its participation in VB100 tests, and was another that submitted a hosted solution. Like most hosted solutions, *Webroot* does a lot more than simply filtering spam – other functions include the provision of business continuity and scanning of email for pornographic images. In an era in which more and more spam is sent from compromised legitimate machines, it is also reassuring to see that the product can be configured to scan outbound messages.

A decent web interface gives system administrators a good overview of current spam and virus threats, as well as an indication of which users are most affected. Unfortunately, due to the way in which the product was set up for this test, few of the options in the interface could be tried out.

In fact, *Webroot*'s developers are already working on finding a way to make the product fit into the test better: a false positive rate of over 25% of all ham messages is almost certainly a sign of product misconfiguration. With



	True negatives	False positives	True positives	False negatives	SC rate	FP rate	FP rate as percentage of total mail corpus
BitDefender Security for Mail Servers	1,652	25	20,478	3,842	84.20%	1.49%	0.096%
ClamAV signatures	1,677	0	6,719	17,601	27.63%	0.00%	0.000%
Giacom	1,624	53	23,470	850	96.50%	3.16%	0.204%
M+Guardian	1,639	38	23,062	1,258	94.83%	2.27%	0.146%
SpamAssassin	1,659	18	14,934	9,386	61.41%	1.07%	0.069%
Webroot E-Mail Security SaaS	1,239	438	23,729	591	97.57%	26.12%	1.685%
Average					**77.02%**	**5.68%**	**0.367%**

such a high false positive rate no certification was awarded this time around, but the developers will no doubt be working hard to achieve significantly better results in the next test.

AWARDS

It cannot be emphasized enough that, in our tests, it is not so much the *absolute* performance of a product that matters, but the *relative* performance compared to that of its competitors. Products will therefore not achieve certification by blocking 'x%' of all spam or generating less than 'y%' false positives. The best-performing products in each test are awarded with one of three certifications:

- VBSpam Platinum for products with a 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 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 spam catch rate and false positive rates are no more than 50% worse than the average in the test.

In this test, based on an average spam catch rate of 77.02% and an average false positive rate of 5.68%, the benchmarks were as follows:

Platinum: SC 88.51%; FP 2.84%

Gold: SC 77.02%; FP 5.68%

Silver: SC 65.53%; FP 8.52%

One does not need a qualification in statistics to understand that these averages have been skewed by the performances of *ClamAV* (which had a very low spam catch rate) and *Webroot* (which had a very high false positive rate). It would thus be tempting to ignore these products when computing the average score. However, we have decided against this based on the fact that we think it is important to stick to the same rules for the duration of a test, rather than change them halfway through.

We are looking into ways in which any future 'outliers' can be excluded from the calculation of averages using non-arbitrary methods.

CONCLUSIONS AND IMPROVEMENTS

If there were two changes I could make to improve the test they would be:

- The inclusion of more products in the test.
- The use of a larger and more varied spam corpus.

Happily, thanks to a great deal of interest from vendors, we anticipate that the number of products participating in the next test (due to be run in June) will reach double figures.

To increase the size of the spam corpus and the variation within it, we intend to work together with *Project Honeypot* – an initiative that has generated the largest and most varied spam trap in the world. The brains behind *Project Honeypot* have kindly offered to relay some of the millions of spam messages they receive to our servers, so that they can be used in our test in real time. This will significantly increase the robustness of the test.

Overall, despite a couple of bugs the first 'live' anti-spam test has been a success, with some encouraging results for most of the participants and a little more work to be done by some of the others. I look forward to the next test to see the effects of a larger field of competition and a larger spam corpus.

Developers interested in submitting products for the next test should contact martijn.grooten@virusbtn.com. The next test will be run during June, with the deadline for product submissions towards the end of May.