JULY 2013

virus BULLETIN

Covering the global threat landscape

CONTENTS

IN THIS ISSUE

SOUNDS RANDOM

The W32/Mammer virus attempts to record ambient sound as a source of true random numbers. Peter Ferrie explains how it does so – and why the implementation is flawed.
page 7

APK ANALYSIS

With a wide variety of e-crime-related threats being discovered every day for Android, the analysis of suspect Android packages is becoming an ever more important task for security researchers. John Foremost introduces Apktool, a very powerful freeware tool for analysing APK files.
page 19

JOURNEY INTO ETHICS

In the latest of his 'Greetz from Academe' series, highlighting some of the work going on in academic circles, John Aycock looks at the thorny issue of ethics in academic security research.
page 22

ISSN 1749-7027

vb

'A series of new factors ... are placing unprecedented evolutionary pressure on the virus/anti-virus/operating system triad.'

Catalin Cosoi, Bitdefender

ON THE CUSP OF EVOLUTIONARY CHANGE

'*For an evolutionary system, continuing development is needed just in order to maintain its fitness relative to the systems it is co-evolving with.*' - L. van Valen (1973)

This oft-quoted phrase is the canonical formulation of the Red Queen theory. When faced with evolving competitors, it says, a species must change at the same pace just to remain within its niche.

This sounds like a recipe for increasing fitness for all participants in the race. Until, that is, you realize that (this being the *Looking Glass* world, after all) all runners do not have to run in a particular direction – it's quite possible to develop (or over-develop) characteristics that help deal with a particular competitor yet which have a net negative survival value. Species can evolve themselves out of existence altogether, racing to an evolutionary dead end.

To complicate matters, the virus vs. anti-virus arms race is in fact a three-way match. We can think of the virus as purely parasitic, the anti-virus as a mutualistic symbiont, and the OS as the host organism.

The three compete for system resources – processing power, memory, bandwidth and data – using two main strategies. They try to wipe out competition outright and make better use of existing resources. Of course, there is intra-phylum competition as well – various flavours of anti-virus, innumerable strains of viruses, and an increasing, but still comparatively small, variety of operating systems.

Of the three, only viruses and anti-virus programs are locked in a predator-prey arms race. And both are obligate guests – neither can function in the absence of a host OS. Meanwhile, operating systems can do just fine on their own, at least as long as they remain fit for purpose.

The ecological equilibrium is established around a position where operating systems (and their designers and users) find it cheaper, in evolutionary terms, to deal with the resource consumption of both virus and anti-virus than to develop better defences of their own. Indeed, overly secure operating systems tend to stray from the equilibrium point by making poor use of resources, while insecure systems lose control of resources altogether, again preventing users from doing useful work.

The anti-virus maintains its niche by being more efficient at fighting threats than its host could ever hope to be without actually losing fitness. The virus uses stealth to hide from its predator and improvements in resource efficiency to avoid smothering its host.

Evolutionary biology also teaches that such equilibria tend to be long-lived, with the actor species random-walking inside their respective phenotypic ranges of variation – for instance, a predator species' fangs might get longer, then shorter, then longer again.

A well-supported theory, however, holds that such dynamic equilibria are punctuated by short periods of massive evolutionary change, quickly reaching a new equilibrium as new species replace the old.

We may be on the cusp of such a change right now. A series of new factors – migration of software services to the cloud, increasing use of encryption for software authentication, and security and hardware virtualization technologies, to name but a few – are placing unprecedented evolutionary pressure on the virus/anti-virus/operating system triad.

The Obad trojan provides a clear-cut example of such co-evolution. This piece of *Android* malware made use of a previously unknown bug in the operating system to make itself impossible to remove from infected systems. Naturally, as anti-virus software on *Android* only has the same permissions as other user programs, almost all anti-virus programs were incapable of removing it. Then, of course, anti-virus makers found a work-around and *Android* OS was patched. The story is ongoing, and it remains to be seen what tricks the next iteration of Obad will come up with.

NEWS

NO MORE LINUX FOR AVIRA

German anti-virus firm and long-standing VB100 regular *Avira* has announced that, as of June 2016, it will no longer offer a *Linux* product.

A statement on the company's website explained that *Avira* is focused on the consumer and micro/small business markets – both of which almost exclusively run *Windows* or *Mac* operating systems, with *Linux* installations declining steadily for several years.

Avira did not enter a product for this year's VB100 test on the *Linux* platform, but gained VB100 certification for all of its *Linux*-based entries between 2004 and 2012.

The company ended active sales and development of its *Linux* products on 30 June, but will continue to deliver detection updates for current users of the product line until June 2016.

ACADEMIC CENTRES OF EXCELLENCE

The University of Cambridge has become one of the latest academic institutions to be recognized as an Academic Centre of Excellence in Cyber Security Research (ACE-CSR) by the UK Government. The well-respected security research group within the University's Computer Lab focuses on topics that include: securing global infrastructure; operating system security; secure computer architectures; network protocol security; security of mobile devices; password authentication; modelling frauds and scams; and protecting location and social network privacy.

The aim of the national scheme to identify cybersecurity centres of excellence is to strengthen the links between the institutions involved in cybersecurity research and the organizations (businesses, government etc.) that could benefit directly from it. Since the scheme was launched last year, 11 institutions have been recognized: Imperial College; Lancaster University; Newcastle University; Queens University Belfast; Royal Holloway, University of London; University College London; University of Birmingham; University of Bristol; University of Cambridge; University of Oxford and University of Southampton.

Two of the institutions that earned recognition last year also recently won a bid to set up Centres for Doctoral Training in Cyber Security. It was announced in May that the University of Oxford and Royal Holloway, University of London will each receive a grant of nearly £4 million from the UK's Engineering and Physical Sciences Research Council (EPSRC) and the Department for Business, Innovation and Skills to host new Centres for Doctoral Training (CDT) in cybersecurity. The government hopes to address the national need for cybersecurity expertise by boosting the number of PhD graduates with relevant skills.

Prevalence Table – May 2013[1]

Malware	Type	%
Adware-misc	Adware	7.92%
Autorun	Worm	7.66%
Java-Exploit	Exploit	6.55%
Heuristic/generic	Trojan	4.58%
BHO/Toolbar-misc	Adware	4.43%
Heuristic/generic	Virus/worm	3.56%
Crypt/Kryptik	Trojan	3.50%
Potentially Unwanted-misc	PU	3.44%
Dorkbot	Worm	3.34%
Iframe-Exploit	Exploit	3.30%
Agent	Trojan	3.09%
Conficker/Downadup	Worm	2.97%
Sirefef	Trojan	2.32%
Sality	Virus	2.29%
Bundpil	Worm	1.76%
Wintrim	Trojan	1.75%
LNK-Exploit	Exploit	1.67%
Downloader-misc	Trojan	1.49%
Zbot	Trojan	1.47%
bProtector	Adware	1.40%
Gamarue	Worm	1.39%
Encrypted/Obfuscated	Misc	1.30%
Ramnit	Trojan	1.26%
Virut	Virus	1.19%
Tracur/Xulcache	Trojan	1.12%
Dropper-misc	Trojan	1.08%
AutoIt	Trojan	0.88%
Injector	Trojan	0.87%
Exploit-misc	Exploit	0.85%
Brontok/Rontokbro	Worm	0.82%
Somoto	Adware	0.75%
Scrinject	Trojan	0.73%
Others[2]		19.28%
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

ALIPIME MAKES A COMEBACK WITH FUJACKS.CB

Ke Zhang
Baidu, China

Alipime [1] is a trojan that monitors web browsing and hijacks online payments by redirecting the user to fake payment pages on certain shopping sites. It was very active in China in 2011, before vanishing for a period of time.

Recently, however, we captured an Alipime threat which was shipped with the W32.Fujacks.CB worm [2], and which utilized a legitimate program to load itself.

INSTALLATION

In order to disguise itself as a legitimate application, Alipime copies itself to the following locations:

- C:\Program Files\ksupdate\360se.exe [Renamed CalendarMain.exe]
- C:\Program Files\ksupdate\sqlite3.dll [Malicious loader]
- C:\Program Files\ksupdate\Resloader.dll [Clean file]
- C:\Program Files\ksupdate\SkinBase.dll [Clean file]

It then creates the following registry entry to make itself persistent on the compromised machine:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\
CurrentVersion\Run]
"Run"="C:\\Program Files\\ksupdate\\360se.exe"
```

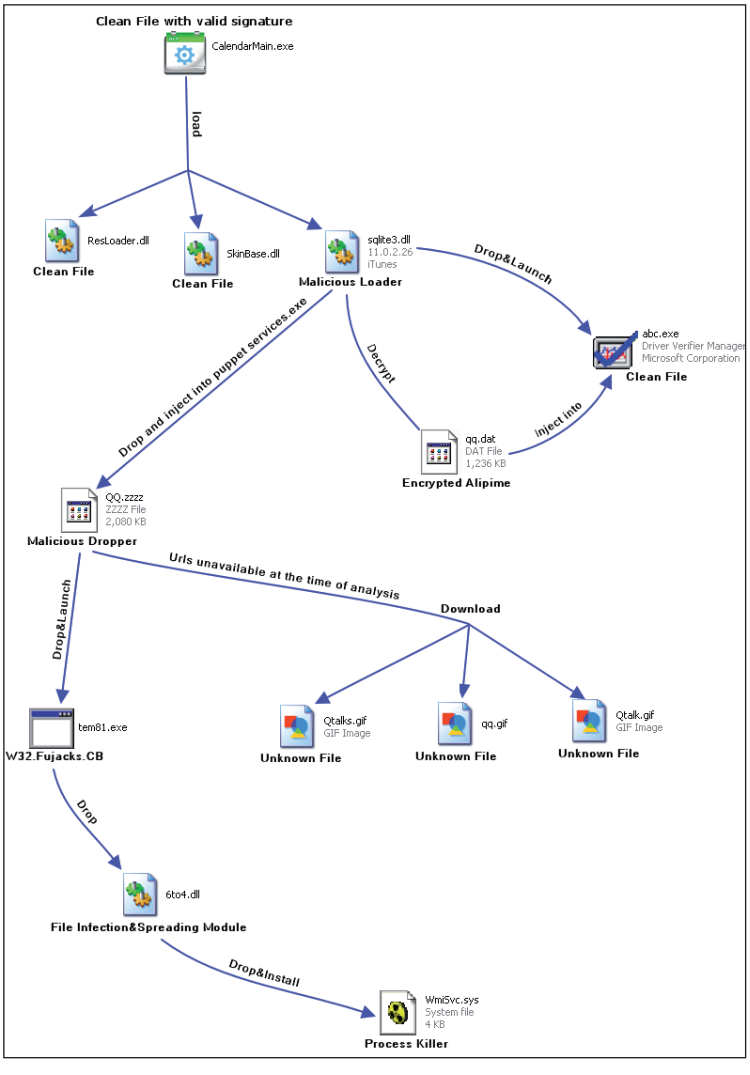


Figure 1: Threat files overview.

DECRYPTION AND INJECTION

The loader uses the RC4 algorithm with the key '1*98$3&^' to decrypt the encrypted Alipime module.

After decryption, the loader will drop a clean file, abc.exe (Microsoft Driver Verifier Manager), and launch it as a puppet, then inject the freshly decrypted Alipime module into it to execute. The loader makes use of the 'process replacement' trick to implement the injection, as the following pseudo code demonstrates:

```
CreateProcess(...,"abc.exe",...,CREATE_SUSPEND,...);
GetThreadContext();
ZwUnmapViewOfSection(...);
VirtualAllocEx(...,ImageBase,SizeOfImage,...);
WriteProcessMemory(...,headers,...);
for (i=0; i < NumberOfSections; i++) {
  WriteProcessMemory(...,section,...);
}
SetThreadContext();
ResumeThread();
```

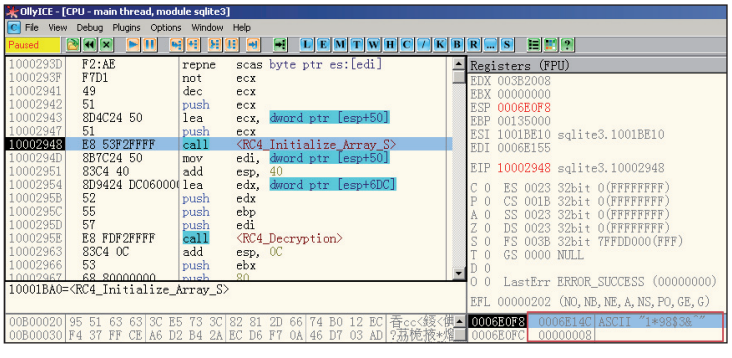


Figure 2: The loader is about to initialize the Array S.

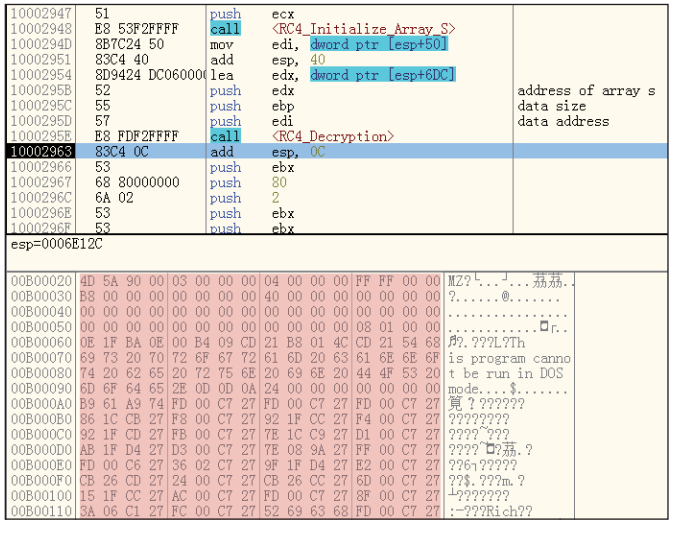


Figure 3: Partially decrypted data.

INTERNET EXPLORER IS DESIRED

Alipime forces the victim to use *Internet Explorer* (*IE*) by eliminating the following processes belonging to other popular web browsers:

sogouexplorer.exe	alisafe.exe	115br.exe
firefox.exe	taobrowser.exe	baidubrowser.exe
twchrome.exe	360chrome.exe	ruiying.exe
chrome.exe	QQBrowser.exe	ETwoOne.exe
maxthon.exe	TTraveler.exe	theWorld.exe
miser.exe	theworld.exe	COrAl.exe
AliimSafe.exe	liebao.exe	top.exe

It needs to obtain the interface pointer of IHTMLDocument2 prior to monitoring and manipulating the pages viewed by the user.

First, it enumerates all the windows and their child windows to find one named 'Internet Explorer_Server' (only *Internet Explorer* and *IE*-based browsers have this child window, which is why *IE* is targeted).

Then it registers the special *Windows* message WM_HTML_GETOBJECT and sends it to the target window to retrieve the IHTMLDocument2 interface pointer. The sample code is as follows:

```
//parameter hwnd = handle of the child window
//Internet Explorer_Server
//I did not put any error check in this code snippet,
//but I have tested it.
void GetDocInterface(HWND hwnd)
{
 HINSTANCE hInstance = NULL;
 CComPtr<IHTMLDocument2> spDoc = NULL;
 LRESULT lRes;
 UINT uMsg;
 LPFNOBJECTFROMLRESULT pfnObjectFromLresult;
 CoInitialize(NULL);
 hInstance = LoadLibraryW(L"OLEACC.dll");
 uMsg = RegisterWindowMessageW(L"WM_HTML_GETOBJECT");
 SendMessageTimeout(hwnd,uMsg,0L,0L,SMTO_ABORTIFHUNG,
1000,(DWORD*)&lRes);
 pfnObjectFromLresult = (LPFNOBJECTFROMLRESULT)GetPro
cAddress(
 hInstance,"ObjectFromLresult");
 (*pfnObjectFromLresult)(lRes,IID_IHTMLDocument2,0,(v
oid**)&spDoc);
 FreeLibrary(hInstance);
 CoUninitialize();
}
```

PAYMENT HIJACKING

Alipime monitors web browsing by calling the method IHTMLDocument2->get_URL. If it finds that the victim is browsing a fast payment page, it will redirect the browser to a standard payment page.

```
IHTMLWindow2 *spWindow2 = NULL;
VARIANT varRet = {0};
BSTR myscript;
BSTR scripttype;
BSTR current_url;
char *urlbuffer;
spDoc->get_URL(&current_url);
urlbuffer = _com_util::ConvertBSTRToString(current_
url);
SysFreeString(current_url);
if (urlbuffer != NULL)
{
    if (strstr(urlbuffer,"standard/fastpay/
fastPayCashier.htm") != 0)
    {
        spDoc->get_parentWindow(&spWindow2);
        if (spWindow2 != NULL)
        {
    //the order_ID is dynamically generated, a
    //typical id is as follows
    //0524741b62f306b421removed_deliberately
          myscript = _com_util::ConvertStringTo
BSTR("document.write("<script>window.location.
href=\"https://cashier.alipay.com/standard/fastpay/
paymentSwitch.htm?orderId=DynamicllyGenerated&target=
standardPayCashier\"")");
          scripttype = _com_util::ConvertStringToBSTR
("javascript");
          VariantInit(&varRet);
          spWindow2->execScript(myscript,scripttype,&
varRet);
          SysFreeString(myscript);
          SysFreeString(scripttype);
          VariantClear(&varRet);
          spWindow2->Release();
        }
    }
}
```

It then replaces the div section 'J-tabcnt-accountDetail' of the standard payment page to prevent the victim from accessing his *Alipay* (a Chinese online payment service) account balance – see Figures 4, 5 and 6.

If it finds that the victim is logging into an online bank to pay for his purchase(s), Alipime will execute a script which uses an embedded account to buy items of equal value on predetermined third-party websites, and replaces the final

Figure 4: Original div section 'J-tabcnt-accountDetail' in the standard payment page.

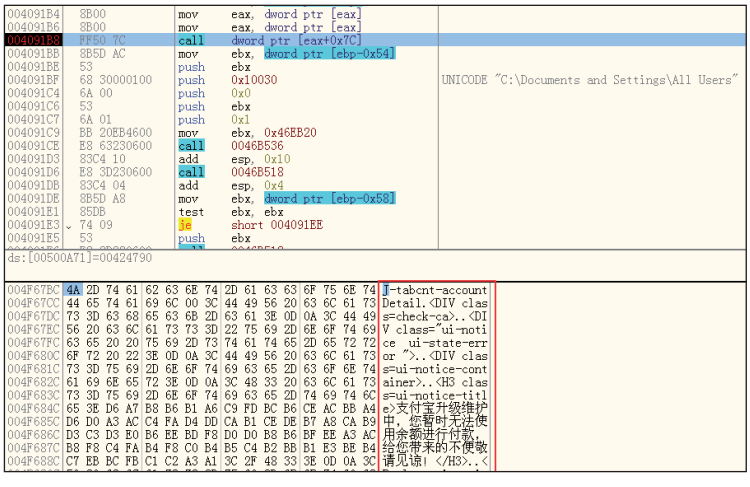


Figure 5: Alipime is about to replace the 'J-tabcnt-accountDetail' div section.

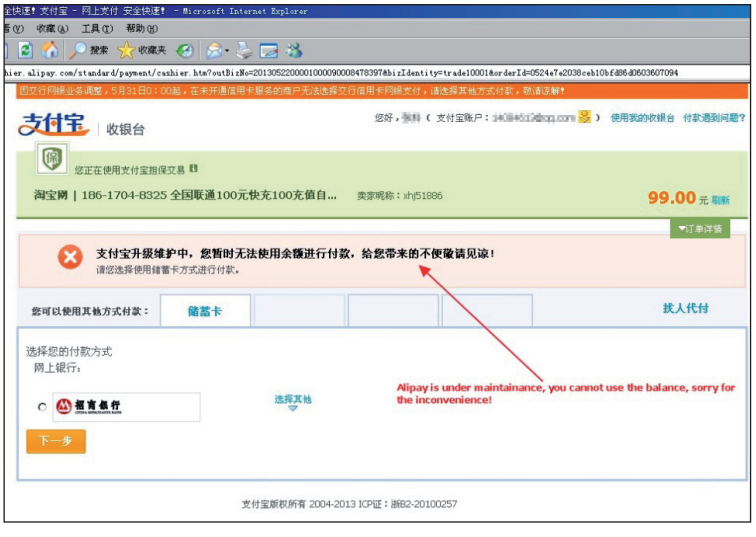


Figure 6: Victim's Alipay account balance becomes unavailable.

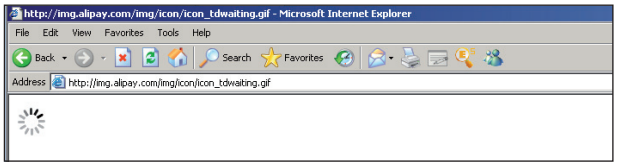


Figure 7: Alipime uses the 'waiting' GIF from Alipay to deceive the victim.

Figure 8: The victim is unable to view his purchase(s).

payment page so that the victim (unknowingly) pays for the attacker's purchases.

In the meantime, Alipime redirects the payment page to 'http://img.alipay.com/img/icon/icon_tdwaiting.gif' to fool the victim into thinking that the unusual time lapse is nothing to worry about (Figure 7).

Alipime also prevents the victim from reviewing his purchase(s) – in this instance by redirecting page 'http://trade.taobao.com/trade/itemlist/list_bought_items.htm' to 'http://trade.taobao.com/trade/confirm_goods.htm?biz_order_id=' so that he cannot find out if he has paid for his purchase(s) successfully (see Figure 8).

CONCLUSION

Alipime makes off with victims' money, but without causing any damage to the infected system – it is a challenge for anti-virus engines to detect and defend against it.

REFERENCES

[1] Trojan.Alipime. http://www.symantec.com/security_response/writeup.jsp?docid=2011-012506-3430-99.

[2] W32.Fujacks.CB. http://www.symantec.com/security_response/writeup.jsp?docid=2009-072015-2415-99.

MALWARE ANALYSIS 2

NOT DROWNING, WAV-ING

Peter Ferrie
Microsoft, USA

There is a big problem with Pseudo Random Number Generation (PRNG) algorithms: they're not random. They require a seed as a starting point, and then generate values in a cyclic manner (of course, the cycle can be very large). Hence, given the seed and the number of iterations, the next value and all subsequent values can be determined. The obvious solution is to make the random number generator really random, by turning the generator into a provider instead, and finding a suitable source of random numbers to collect. The W32/Mammer virus attempts to do just that.

IMPORT BUSINESS

The virus begins by registering a Structured Exception Handler in order to intercept any errors that occur during infection. The virus retrieves the base address of kernel32.dll. It does this by walking the InMemoryOrderModuleList from the PEB_LDR_DATA structure in the Process Environment Block. The address of kernel32.dll is always the second entry in the list. The virus assumes that the entry is valid and that a PE header is present there. This assumption is fine, because of the Structured Exception Handler that the dropper has registered.

The virus resolves the addresses of the API functions that it requires: find, set attributes, open, map, unmap, close, malloc, free and LoadLibrary. The virus uses hashes instead of names and uses a reverse polynomial to calculate the hash. Since the hashes are sorted alphabetically according to the strings that they represent, the export table needs to be parsed only once for all of the APIs. Each API address is placed on the stack for easy access, but because stacks move downwards in memory, the addresses end up in reverse order in memory. The hash table is terminated with a single byte whose value is zero. While this saves three bytes of data, it also prevents the use of any API whose hash ends with that value. This is not a problem for the virus in its current form, since none of the needed APIs have such a value, but it could result in some surprises for any virus writer who subsequently tries to extend the code.

The virus loads 'winmm.dll' and resolves the addresses of the API functions that it requires, using the hash method again. It then allocates a 689KB buffer.

TIDAL WAVE

The virus registers a second Structured Exception Handler, and then opens the audio input device for recording. The

recording format is a *Microsoft*-proprietary audio format, using two channels of 44,100Hz, and 16-bit samples at 172KB/s. The input is not filtered in any way. The virus registers a third Structured Exception Handler, prepares the audio buffer for receiving data, and then initiates the recording. Next, the virus repeatedly calls an API to finalize the buffer, and loops while the function returns a status indicating that the buffer is not yet full.

The reason the virus has to call the API repeatedly is because it did not register a callback function when it prepared the audio buffer. If the virus had registered a callback function, then that function would receive the notification that the buffer was full, and the virus could have used a delay loop to wait for that event to occur.

The virus checks only the low byte of the status value, which might appear to be a bug, but this is actually safe because of the way in which the error codes are constructed for the API. The error codes are built on 'bases', which are specific to the type of function in use. The base that corresponds to the wave format fits entirely within an eight-bit value, so only those eight bits need to be checked.

Once the virus receives the status that the buffer is full, it uses a breakpoint instruction to force an exception to occur and to transfer control to the third Structured Exception Handler. This technique was first seen in the Chiton [1] family, and it appears a number of times in the virus code. It is an elegant way to reduce the code size, in addition to functioning as an effective anti-debugging method. Since the virus has protected itself against errors by installing a Structured Exception Handler, the simulation of an error condition results in the execution of a common block of code to exit a routine. This avoids the need for separate handlers for successful and unsuccessful code completion.

When the third Structured Exception Handler receives control, it closes the audio device, and then checks if the buffer received at least 16 bytes before the exception occurred. If the buffer is not full enough, then the virus uses another breakpoint instruction to force an exception to occur and to transfer control to the second Structured Exception Handler. When the second Structured Exception Handler receives control, it frees the allocated buffer, and then uses yet another breakpoint instruction to force an exception to occur and to transfer control to the first Structured Exception Handler.

MISTYPE, MISS TYPE

If the buffer is full enough, then the virus intends to copy the first 16 bytes from the buffer to use as decryption keys. The one major bug in the virus code is right here. A simple typographical error – the letter 'a' instead of the letter 'c' – results in the wrong buffer being used as the source for the keys. So, instead of copying the audio buffer *data*, the audio buffer *header* is copied. The result is that three of the four keys are entirely constant, and half of the remaining key is constant, too. This makes decryption trivial, even in the absence of an emulator. The fact that the bug was not discovered suggests that the operating system that was used for testing is one that implements Address Space Layout Randomization, since otherwise even the fourth key would very likely be constant. Of course, this observation is simply a minor point of interest and serves no other purpose.

BITS AND PIECES

In any case, the virus begins the replication phase as though everything were fine. It registers a fourth Structured Exception Handler, and then searches in the current directory (only) for PE files, regardless of their extension. The virus uses *Unicode*-only APIs, which allows it to examine files that would otherwise be inaccessible to ANSI APIs. It uses a nice trick to find the files, which was first seen in the Chiton [1] family: the file mask is '*' which, when pushed onto the stack, can be interpreted as a zero-terminated *Unicode* string because it is followed by three zeroes. The rest of the code is derived from the Mikasa [2] virus.

The virus attempts to remove the read-only attribute from whatever is found. It attempts to open the found object and map a view of it. If the object is a directory, then this action will fail and the map pointer will be null. Any attempt to inspect such an object will cause an exception to occur, which the virus will intercept. If the map can be created, then the virus will inspect the file for its ability to be infected.

SEEK AND DESTROY

The virus is interested in Portable Executable files for the *Intel* x86 platform that are not DLLs or system files. The check for system files could serve as a light inoculation method, since *Windows* ignores this flag. The virus checks the COFF magic number, which is unusual, but correct. The reason for checking the value of the COFF magic number is to be sure that the file is a 32-bit image. This is the safest way to determine that fact because, apart from the executable ('IMAGE_FILE_EXECUTABLE_IMAGE') and DLL ('IMAGE_FILE_DLL') flags in the Characteristics field, all of the other flags are essentially ignored by *Windows* (from the point of view of the virus, that is true, but technically it's not quite accurate – setting the 'IMAGE_FILE_RELOCS_STRIPPED' flag has the effect of disabling Address Space Layout Randomization for the process). This

includes the flag ('IMAGE_FILE_32BIT_MACHINE') that specifies that the file is for 32-bit systems.

As an added precaution, the virus checks that the size of the optional header is large enough to hold the BaseRelocationTable directory. If the optional header is also large enough to hold the LoadConfigurationTable data directory, then the virus requires that the LoadConfigurationTable RVA is zero. The reason for this last check is because the table includes the SafeSEH structures, which will prevent the virus from using arbitrary exceptions to transfer control to other locations within its body. The virus checks that the file targets the GUI subsystem.

RELOCATION ALLOWANCE

The virus checks the Base Relocation Table data directory to see if the relocation table begins at the exact start of the last section. If it does, then the virus assumes that the entire section is devoted to relocation information. This could be considered to be too strict. The virus checks that the physical size of the section is large enough to hold the virus code. There are two bugs in this check.

The first bug is that the size of the relocation table could be much smaller than the size of the section, and other data might follow it. The data might be overwritten when the virus infects the file. Further, the value in the Size field of the Base Relocation Table data directory cannot be less than the size of the relocation information, and it cannot be larger than the size of the section. This is because the value in the Size field is used as the input to a loop that applies the relocation information. It must be at least as large as the sum of the sizes of the relocation data structures. However, if the value were larger than the size of the relocation information, then the loop would access data after the relocation table, and that data would be interpreted as relocation data. If the relocation type were not a valid value, then the file would not load. If the value in the Size field were less than the size of the relocation information, then it would eventually become negative and the loop would parse data until it hit the end of the image and caused an exception.

The second bug is that by checking only the physical size and not the virtual size as well, whatever the virus places in the file might be truncated in memory if the virtual size of the section is smaller than the physical size of the section.

TOUCH AND GO

If the section appears to be large enough, then the virus overwrites the relocation table with the decryptor and the encrypted virus body. Overwriting the relocation table means that infected files do not show an increase in file size. The encryption method is to use 32 rounds of XTEA, using the 'keys' from above. The virus changes the section characteristics to writable and executable, and sets the host entry point to point directly to the virus code. The virus clears only two flags in the DLL Characteristics field: IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY and IMAGE_DLLCHARACTERISTICS_NO_SEH. This allows signed files to be altered without triggering an error, and enables Structured Exception Handling. The virus also zeroes the Base Relocation Table data directory entry, to prevent the virus code from being interpreted as relocation data, in the event that the file opted in to Address Space Layout Randomization.

The host's original entry point RVA is saved in the decryptor code. When the decrypted code is run, the virus converts the RVA to a virtual address by adding the ImageBase value from the Process Environment Block to it. This allows the virus to behave correctly if the file is relocated in memory.

Once the infection process has completed, the virus uses a breakpoint instruction to force an exception to occur and to transfer control to the fourth Structured Exception Handler. When the fourth Structured Exception Handler receives control, it unmaps and closes the file, and restores its file attributes, but not the file date and times. After all files have been examined, the virus uses a breakpoint instruction to force an exception to occur and to transfer control to the first Structured Exception Handler. When the first Structured Exception Handler receives control, it transfers control to the host entry point.

CONCLUSION

True random number generation certainly has its uses, and the recording of ambient sound as a source of random numbers is a valid technique, even though the implementation is flawed in this example. In any case, this virus gains no advantage by using a truly random number generator, because it must still carry the decryption keys.

REFERENCES

[1] Ferrie, P. Unexpected Resutls [sic]. Virus Bulletin, June 2006, p.4. http://www.virusbtn.com/pdf/magazine/2002/200206.pdf.

[2] Ferrie, P. It's mental static! Virus Bulletin, March 2013, p.8. http://www.virusbtn.com/pdf/magazine/2013/201303.pdf.

MALWARE ANALYSIS 3

WHO'S BAD? SKYBOT OR NGRBOT

Neo Tan, Christy Chung & Kyle Yang
Fortinet, Canada

The SkyBot and NgrBot (a.k.a. DorkBot) worms are often confused with each other, since their methods of spreading are very similar (both can spread through portable drives or via IM). However, the two are very distinct in terms of the channel of distribution: SkyBot spreads itself/other malware by tricking users into clicking a malicious link that is sent through the *Skype* window, whereas NgrBot spreads by sending a malicious link through *MSN* and by posting a malicious link on social networking sites (*Facebook*, *Twitter*, etc.). In addition, NgrBot is able to download other malware under the instruction of IRC commands.

In this article, we will take a detailed look at these two IM worms – from their hijack methods and distribution channels, to the other malicious files they are trying to deliver – in order to give a brief comparison of the two.

1. SKYBOT

Hijack Skype

The latest SkyBot only targets *Skype*, and spreads itself by sending a malicious link to all the contacts in the victim's *Skype* contact list. In its first phase, it tries to send the malicious link to the active *Skype* chat window. In order to hijack the current active chat window, SkyBot goes through the following steps:

1. It calls FindWindowA to find 'tSkMainForm' or 'tSkMainForm.UnicodeClass' to get the *Skype* window handle, then ShowWindow to make it active.
2. It calls FindWindowExA with the obtained handle and gets the handle of the 'TConversationsControl' window.
3. It calls SetForegroundWindow to bring the chat window into the foreground and activates the window.
4. It calls ShowWindow with parameter SW_RESTORE to activate and display the *Skype* window (so that if the *Skype* window is minimized, the system will restore it to its original size and position).
5. It then sleeps for 100ms, then uses the WM_SETFOCUS parameter in the SendMessageA function to send to the *Skype* window in order to gain the keyboard focus.
6. It calls SendMessageA with the WM_KEYDOWN parameter to simulate the 'Up Arrow' key event 0x320 times, and the 'Down Arrow' key event 0x2 times, followed by an 'Enter' key event.
7. It calls BlockInput to block any user input, then enters into a loop. The loop will break if the current active chat user receives the malicious link.
8. It finds the 'TConversationForm' using the FindWindow API or, if it is not successful, it tries to find 'tSkMainForm' instead and EnumChildWindows to find its child windows. Then it finds 'TChatEntryControl', then 'TChatRichEdit', which is the text input field of the *Skype* chat window.
9. It calls SendMessageW with the WM_SETTEXT parameter to fill in the malicious link and sends the 'Enter' key event to finish the sending process. Then the loop breaks and user input is unblocked.

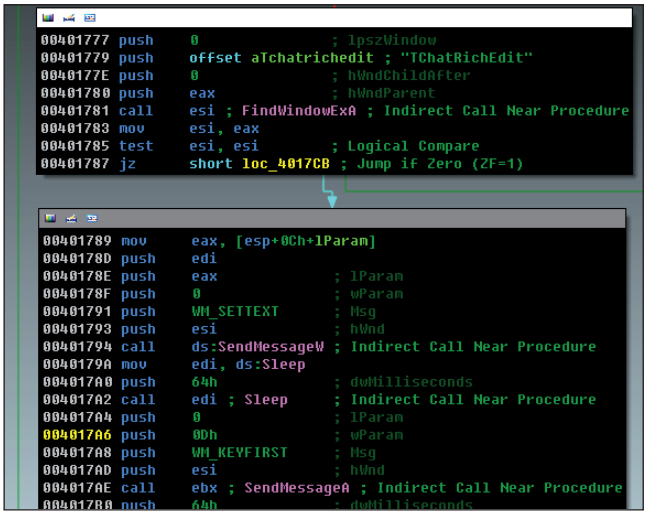


Figure 1: SendMessage sets the spam text and sends an 'Enter' key event (virtual-key code: 0Dh).

The second phase is to send the malicious link to the entire list of the user's contacts, regardless of whether they are online or offline. After hijacking the current active chat window to send the malicious link, it sleeps for 0x4e20 seconds and then uses the *Skype* Desktop API (also called *Skype* public API) provided by *Skype* itself. To do that, it utilizes the Skype4COM.dll file, which comes with the installation of every *Skype* application. Skype4COM is a *Windows*-based COM object which simply bridges the text-based *Skype* Desktop API to a third-party application. To import the Skype4COM.dll, it calls CoCreateInstance with the hard-coded rclsid.

Then it creates the wrapper object and calls ISkype.Attach with Protocol Version = 8 and Wait = -1. Although the

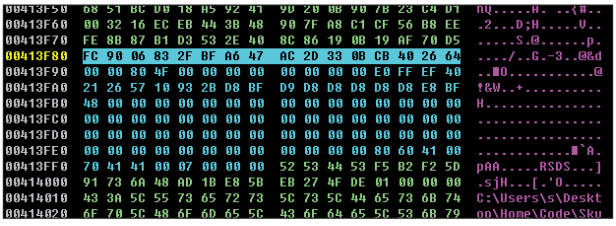


Figure 2: clsid:830690FC-BF2F-47A6-AC2D-330BCB402664 hard-coded in the worm.

Wait parameter is set to -1 here, this Attach method will still trigger the *Skype* application to warn the user that an application is trying to use *Skype*, and will let the user decide whether to allow it. However, this worm uses a trick to bypass the warning (this technique still works on the latest version of *Skype* [6.3.0.105] at the time of writing this article):

Before calling the Attach, it creates a thread which watches for warning windows popping up in *Skype*. The main idea here is to find the 'TZapCommunicator' window. First, it tries to call FindWindowA with ClassName 'tSkMainForm' or 'tSkMainForm.UnicodeClass'. (The 'TCommunicatorForm' window will be searched if the above two searches fail – this is probably for backward-compatibility with older versions of *Skype*.) If there is a hit, it uses the return handle to call FindWindowExA to search for its child window, 'TZapCommunicator'. The search is done in a loop, with a 0xc8 millisecond sleep. If it finds the 'TZapCommunicator' window, it checks the foreground window by calling GetForegroundWindow; it will minimize the current foreground window if it is not the *Skype* window and then set the *Skype* window as the foreground window by calling SetForegroundWindow. This step is crucial for the later part. If for some reason the foreground window is not *Skype*, the latest bypass method will fail.

It calls GetWindowRect to get the position of the 'TZapCommunicator' window and calculate the size. Then it calls GetSystemMetrics twice with index equal to SM_CXSCREEN and SM_CYSCREEN to obtain the primary screen's width and height in pixels. Then it calculates the absolute position of the 'allow' button in pixels. The SendInput API is called three times, with the type set to INPUT_MOUSE: the first SendInput API moves the mouse to the absolute position of the 'allow' button, the second sends a mouse left-button-down event, and the third sends a mouse left-button-up event to finish the job.

Once the attaching is successful, it uses ISkype.Friends to retrieve the victim's list of contacts and iterate through them one by one to send the malicious link using ISkype.SendMessage.

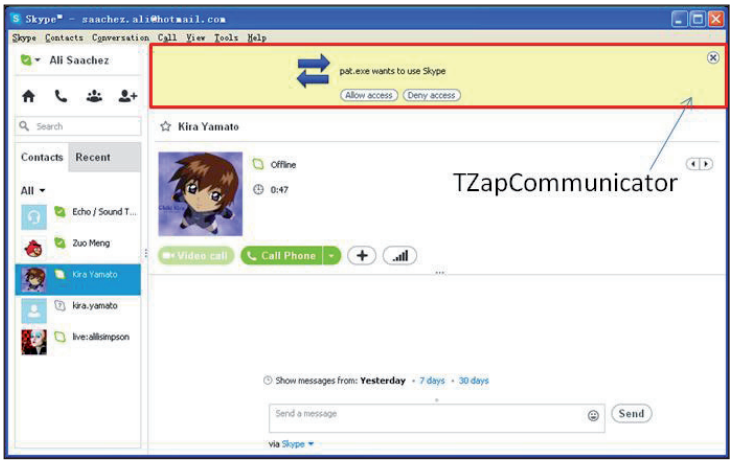


Figure 3: Attach usually triggers a warning to the user.

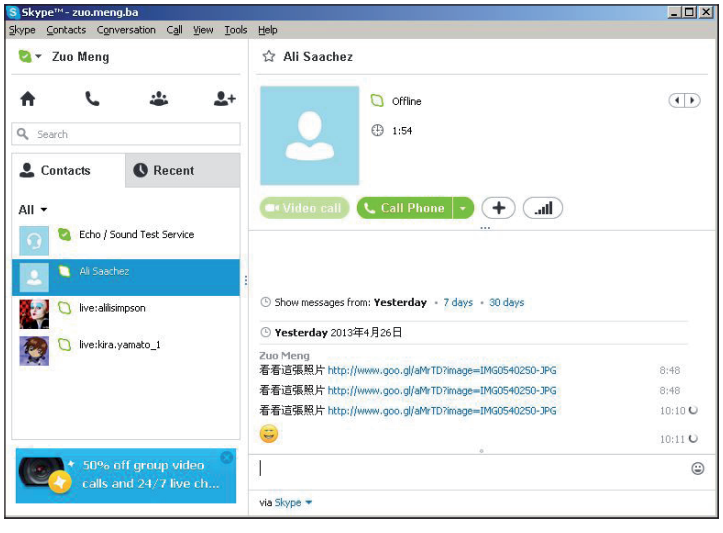


Figure 4: Spam is sent to all contacts.

Payload

The infected link leads to an IRCBot which eventually will download a type of *Bitcoin* miner (a CPU-based miner). This miner only uses the CPU to perform mining, thus it always utilizes about 90% of the CPU resources when running (which is not an efficient method compared with other, GPU-based miners).

2. NGRBOT

The NgrBot worm is able to spread through portable drives, social networks and IM (but not *Skype*).

Once the NgrBot has been installed on the computer, the malware injects the IRC communication routine into a

newly created mspaint.exe process. Then it connects to the IRC server to get commands. Figure 5 shows a screenshot of a typical communication in the IRC channel.

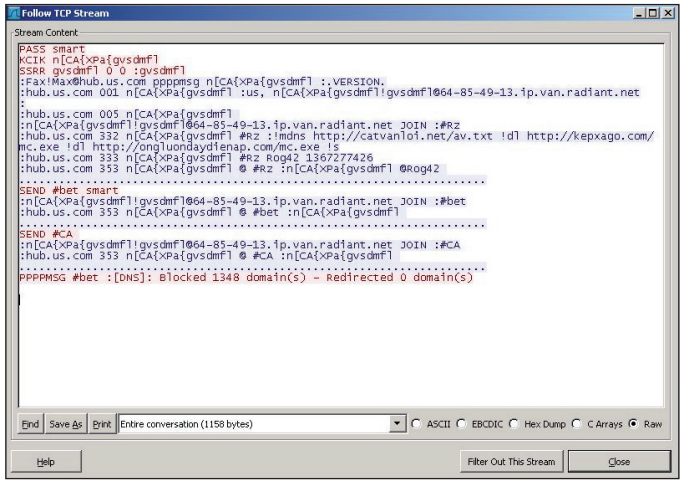


Figure 5: Communication between the IRCBot and the server.

From Figure 5, we can see that the NgrBot joins the IRC channel with a formatted 'nickname' ('n[CA{XPa{gvsdmfl' in this case). The nickname is based on the following template:

'n[Country Code{OS VersionlUser Type{Random String'

Where User Type = 'a' means that it is an administrator, and User Type = 'u' means that it is a normal user.

After it joins the IRC channel, the server issues the command '!mdns' with a link to a DNS block list and the command '!dl' with two download jobs. Note that these two commands are not fixed; they can be fully customized via the NgrBot builder.

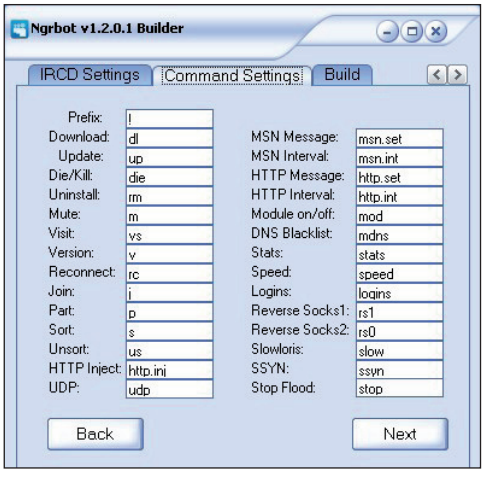


Figure 6: Command setting of NgrBot Builder.

From Figure 6, we can see that the IRC server can send a fake message to be used by browsers and IM, hook APIs to hijack user messages, send a list of anti-virus vendor domain names (most of which are anti-virus application update server URLs) and a table of redirecting DNS queries to be used by the hooked DNS query APIs, receive stolen user account information, infect removable drives, download and execute files to/from the bot, and perform distributed denial of service attacks, etc.

Inline hook

After blocking the domains, it injects itself into almost every process (except mspaint.exe) that it is able to access, so that it can look for and hook the desired APIs via inline hooking. The desired API list is hard-coded in the bot's binary, as shown in Figure 7. The data structure of each entry in the hooking list is described as follows:

```
struct API_TO_HOOK
{
        DWORD *libName;
        DWORD *apiName;
        DWORD *hookerFunction;    //function to be
injected into the original API routine
        DWORD *outRestorepoint;   //inline-hook
restore point
        DWORD *browserMessengerCheckingFunction;
//helper function to check if current process is a
browser or a messenger
}
```

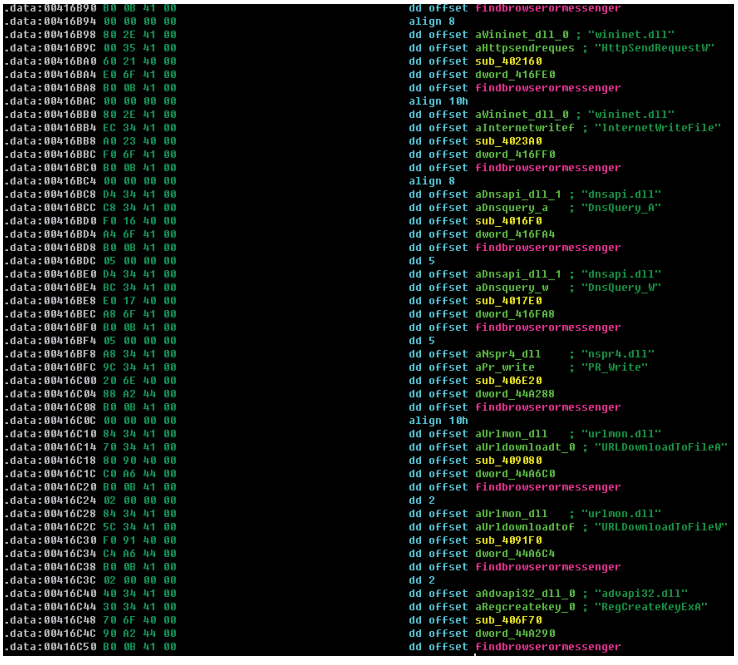


Figure 7: List of hooking APIs.

CopyFileA	CopyFileW	CreateFileA	CreateFileW	DeleteFileA
DeleteFileA	DeleteFileW	DnsQuery_A	DnsQuery_W	GetAddrInfoW
HttpSendRequestA	HttpSendRequestW	InternetWriteFile	MoveFileA	MoveFileW
NtEnumerateValueKey	NtQueryDirectoryFile	PR_Write	RegCreateKeyExA	RegCreateKeyW
send	URLDownloadToFileA	URLDownloadToFileW		

Table 1: All the targeted APIs.

Webroot	Fortinet	Virusbuster.nprotect	Gdatasoftware	Virus
Precsesecurity	Lavasoft	Heck.tc	Emisoft	Onlinemalwarescanner
Onecare.live	f-secure	Bullguard	Clamav	Panadasecurity
Sophos	Malwarebytes	Sunbeltsoftware	Norton	Norman
McAfee	Symantec	Comodo	Avast	Avira
Avg	Bitdefender	Eset	Kaspersky	Trendmicro
Iseclab	Virscan	Garyshood	Viruschief	Jotti
Threatexpert	Novirusthanks	Virustotal		

Table 2: Keywords contained in the hard-coded DNS blocking list.

The libName is a pointer to the name of the library which contains the API. The apiName is a pointer to the name of the targeted API. The hookerFunction is the address of the malicious function which will hook the original API. And the outRestorepoint is the pointer to a pre-determined location which holds a copy of the overwritten byte codes during the inline hooking and a jump operation back to the original API work flow.

Table 1 shows all the targeted APIs.

Among them, the hooking of CopyFileA, CopyFileW, CreateFileA, CreateFileW, DeleteFileA, DeleteFileW, MoveFileA, MoveFileW, NtEnumerateValueKey, NtQueryDirectoryFile, RegCreateKeyExA and RegCreateKeyW are mainly for the bot's self-defence mechanism. If any other process attempts to access the bot's registry record, it will block it. The hooking of DnsQuery_A, DnsQuery_W and GetAddrInfoW is for the blocking or redirecting of DNS queries. Table 2 shows the keywords contained in the hard-coded DNS blocking list.

After matching this list, it will access the shared data through a named pipe to see if there is a downloaded domain name list, and block those domains as well.

The hooking of HttpSendRequestA, HttpSendRequestW, InternetWriteFile, PR_Write and send is for accessing and modifying the user's browser or messenger communication, so that it can grab sensitive information and also hijack messages. However, it only has the ability to parse *MSN* messenger protocol, so *Skype* is safe from this worm for now.

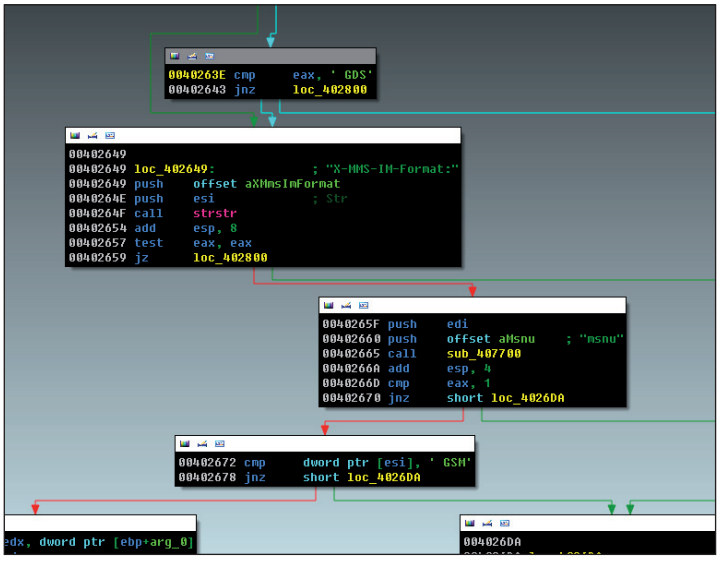


Figure 8: The hooker function tries to match MSN protocol keywords in the message.

Table 3 shows a list of websites from which it tries to grab login information.

The hooking of URLDownloadToFileA and URLDownloadToFileW blocks the downloading of any file with extension 'exe', 'com', 'pif' or 'scr' from the *IE* or *Firefox* browser. It does not affect the bot's downloading process since the bot does not use browsers to do that. This

4shared	Alertpay	AOL	Bcointernacional	BigString
Brazzers	cPanel	Directadmin	Dotster	DynDNS
eBay	Enom	Facebook	Fastmail	Fileserve
Filesonic	Freakshare	Gmail	GMX	Godaddy
Hackforums	Hotfile	IKnowThatGirl	Letitbit	Live
LogMeIn	Mediafire	Megaupload	Moneybookers	Moniker
Namecheap	Netflix	Netload	NoIP	OfficeBanking
Oron	Runescape	Sendspace	Sms4file	Speedyshare
Steam	Thepiratebay	Torrentleech	Twitter	Uploaded
Uploading	Vip-file	Webnames	Whatcd	WHM
Yahoo	YouTube	YouPorn		

Table 3: Websites from which it tries to grab login information.

is some kind of self-defence from competitors, against any other possible exploits.

The concept of this inline hooking is to replace the beginning of the API calls with a jump to the malicious code, and after executing the malicious code, the flow jumps to the saved original opcodes, then finally jumps back to resume from the original location. The malware uses a method called 'code overwriting' to locate the address of the original API function, and changes the first five bytes of the API code with a unconditional Jump instruction that redirects the call to the callback function. The following is an example of how it hooks the kernel32.CreateFileA API:

1. Check the import tables of each and every DLL against the hooking APIs list, and get the addresses of the function to hook.
2. Calculate the offset to the malicious hooker function.
3. Replace the original code with a jump (0xE9) and the hooker function distance.

```
0093110C 7C809B32 kernel32.VirtualFreeEx
00931110 7C801A24 kernel32.CreateFileA
00931114 7C809C6E kernel32.WaitForMultipleObjects
```

```
7C801A22    90          nop
7C801A23    90          nop
7C801A24  - E9 97F71184 jmp 009211C0
7C801A29    FF75 08     push dword ptr ss:[ebp+8]
7C801A2C    E8 73C80000 call kernel32.7C80E2A4
```

Figure 9: Inline hooked CreateFileA.

4. Store the replaced byte codes at a pre-defined location, and append a jump back to 0x7C801A29.

Named pipe

In order to pass the commands to the injected processes, it implements the named pipe technique for the communication between the IRC function process and the worker processes. The data saved in the pipe is RC4 encrypted; the RC4 key is hard-coded in the binary and its CRC32 value is used to compose the pipe name.

```
0040A38E lea     ecx, [ebp+String]
0040A394 push    offset aS_0     ; "%s"
0040A399 push    ecx             ; Dest
0040A39A call    sprintf
0040A39F mov     edi, ds:lstrlenA
0040A3A5 add     esp, 40h
0040A3A8 push    offset rc4key   ; lpString
0040A3AD call    edi ; lstrlenA
0040A3AF push    eax
0040A3B0 push    offset rc4key
0040A3B5 call    crc32
0040A3BA push    eax
0040A3BB push    offset a_Pipe08x_ipc ; "\\\\.\\pipe\\%08x_ipc"
0040A3C0 lea     edx, [ebp+FileName]
0040A3C6 push    3FFh            ; Count
0040A3CB push    edx             ; Dest
0040A3CC call    _snprintf
0040A3D1 add     esp, 18h
0040A3D4 lea     eax, [ebp+String]
0040A3DA push    eax             ; lpString
0040A3DB call    edi ; lstrlenA
0040A3DD mov     esi, eax
```

Figure 10: The pipe name is in the following format: [Hex value of CRC32]_ipc.

The named pipe server is created in the mspaint.exe thread, and another process calls ConnectNamedPipe using the same CRC32 value as the name to get the pipe handle. Then they are able to access the shared data just like a local file object.

Once the above set-ups are finished, NgrBot copies itself to the current user's %AppData% folder with a randomly generated name and then adds a link to its executable file in the system registry autorun key in order to automatically launch each time *Windows* starts up.

Payload

The download of the latest NgrBot is also a *Bitcoin* miner, but this one is a lot more sophisticated than the CPU miner downloaded by SkyBot. It imports the OpenCL library and utilizes GPUs to do the mining, which is much more efficient than a CPU-based miner.

COMPARISON OF TWO IM WORMS

	SkyBot	NgrBot
Hijack method	- *Windows* hijack - Use of Skype4COM library	Inline hooks
Spreading method/targets	Sends spam with infected URL to messenger via *Skype* API	-Hijacks *MSN* messages -Uses stolen social media accounts to send spam
C&C control method	IRC*	IRC
Infects removable drives?	No†	Yes
Persistent?	No‡	Creates autorun entry in registry
PPI	- IRCBot - *Bitcoin* miner (CPU-based miner)	- *Bitcoin* miner (GPU-based miner) - Kelihos

*The *Skype* spam leads to IRCBot download.
†Its IRCBot can infect removable drives.
‡Its IRCBot creates an autorun entry in the registry.

CONCLUSION

Compared with NgrBot, SkyBot is more likely to be a spreading module of its IRCBot, with the IRCBot in charge of its updates and persistency. In terms of spreading through messengers, SkyBot only targets *Skype* and NgrBot targets *MSN* messenger – which will soon be retired following *Microsoft*'s acquisition of *Skype*. Because *Skype* is a multi-platform messenger, if the spam message directs the user to a web page which can dynamically generate redirects to different payloads according to the detected user platform, this worm has the potential to spread through other operating systems such as *Android* and *iOS*. On the other hand, the payloads of both worms currently lead to *Bitcoin* miners. This is not a curious coincidence since the value of *Bitcoins* is increasing rapidly in the Internet society. Both worms use IRC commands to communicate with C&C servers, and their traffic is virtually unencrypted – thus making detection very easy.

MALWARE ANALYSIS 4

UNLOCKING LOCKSCREEN

Walter (Tiezhu) Kong & Kyle Yang
Fortinet, Canada

LockScreen is characterized by a piece of malware that locks the victim's screen, displays the logos of a police department or law enforcement agency and accuses the victim of having committed an offence. To unlock the computer, the victim is asked to pay a small fine through an electronic payment service such as *Paysafecard*, *Ukash*, etc. It may also impersonate an officially recognized organization in charge of collecting taxes or various fees, or even a fictitious organization that claims to be responsible for collecting such payments. Figure 1 shows an example.

Figure 1: Example of LockScreen.

During our analysis of LockScreen, we found that this piece of malware uses more anti-debug tricks than a lot of the malware we usually see. In this article, we will detail all the anti-debug tricks before shedding light on the communication protocol and encryption algorithm between the bot and the C&C server.

ANTI-DEBUG TRICKS

1. Kernel API modification

The first anti-debug trick we faced was 'kernel API modification'. In this case, LockScreen redirects its code

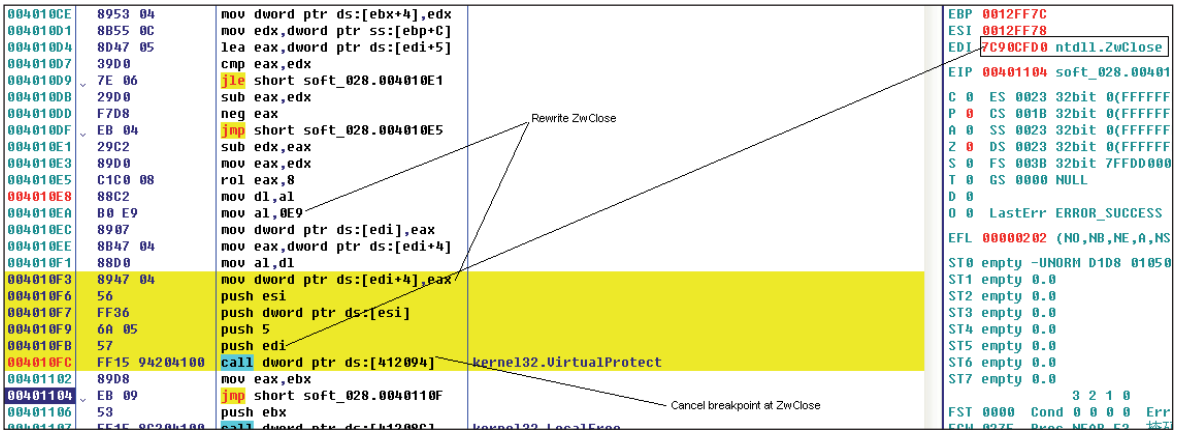


Figure 2: ZwClose modification routine.

flow by modifying the ZwClose API. First, it gets the ZwClose API by calling the GetProcAddress function, then it backs up the first five bytes of ZwClose to a specific location for future restoration. Finally, it adds an unconditional jump which leads the code flow back to itself. Figure 2 shows the ZwClose modification routine, and Figure 3 shows the modified ZwClose function.

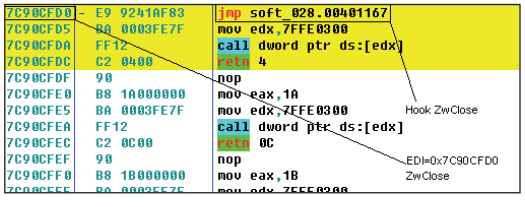


Figure 3: Modified ZwClose API.

After the modification, it calls the LocalFree API, which eventually calls the modified ZwClose API.

2. Debug port check via ZwQueryInformationProcess API

LockScreen will determine whether a debugger is attached to a process by calling the ZwQueryInformationProcess API with the ProcessInformationClass parameter set to 7 (ProcessDebugPort). This API will query the DebugPort field of the EPROCESS kernel structure. A non-zero value in the DebugPort field indicates that the process is being debugged.

3. Hide thread from debugger via ZwSetInformationThread API

The ZwSetInformationThread is usually used for setting a thread's priority. However, it can be used to prevent a debugging event from sending from the kernel function bgkpSendApiMessage(). Figure 5 shows an example of such a technique.

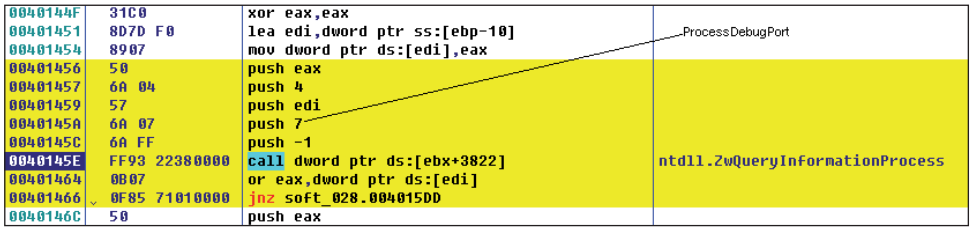


Figure 4: Check debug port.

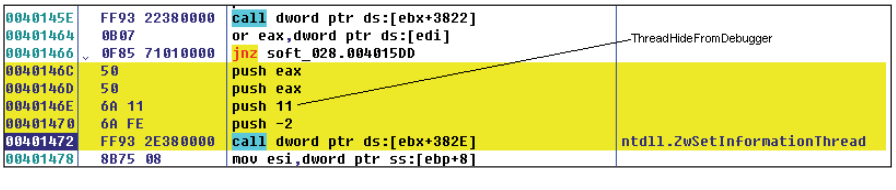


Figure 5: Hide thread from debugger.

In Figure 5, ThreadHideFromDebugger (0x11) is set to the ThreadInformationClass parameter when calling ZwSetInformationThread, which results in the ThreadHideFromDebugger setting the HideThreadFromDebugger field of the ETHREAD16 kernel structure. Therefore, the send event to the debugger function is never invoked.

4. Hide all via the SwitchDesktop API

Windows NT-based platforms support multiple desktops, and it is possible to select a different active desktop. LockScreen will create a desktop by calling CreateDesktop with GENERIC_ALL set to dwDesiresAccess. After that, it invokes the SwitchDesktop API (Figure 6), which results in the OS switching to a new desktop which is used for LockScreen to display the scam screen. It will hide the previously selected desktop, with no obvious way to switch back to the old one with our debugger on it.

COMMUNICATION PROTOCOL AND ENCRYPTION ALGORITHM

Injection

LockScreen first lands in the explorer.exe process via injection, then it injects its core code into the svchost.exe process. The injected code creates three threads which kill the taskmgr.exe process, create an autorun registry entry and copy the malware to a specific location. After that, it tries to use the SwitchDesktop API in a loop as a final barrier to the communication routine (see Figure 7).

Preparing communication data

To prepare the communication data, LockScreen first retrieves the computer name and turns it into a hash by using a custom algorithm. Then it gets the C:\ VolumeSerialNumber by calling the GetVolumeInformation API. Finally, it retrieves the OS minor version by calling the GetVersionEx API. All of the gathered information will form into the structure shown in Figure 8.

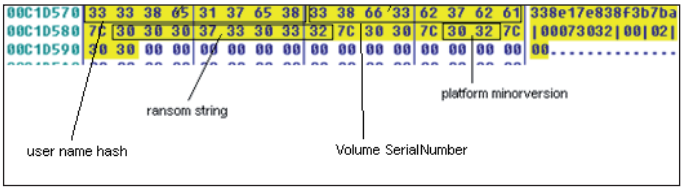


Figure 8: Basic computer information.

Next, it encrypts the above data using the RC4 algorithm with the result of the current time-stamp multiplied by 0x18D as the key (QWORD). After the encryption, it will prepend the key to the encrypted data.

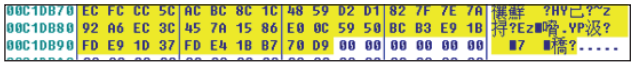


Figure 9: Encrypted basic computer information.

Generate C&C URL

In order to generate the C&C server domain name and URL, it will decrypt data from a specific location in the sample using the following algorithm:

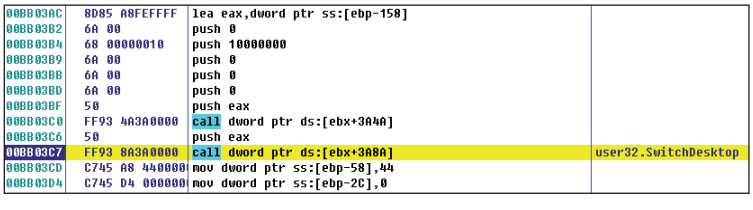


Figure 6: SwitchDesktop.

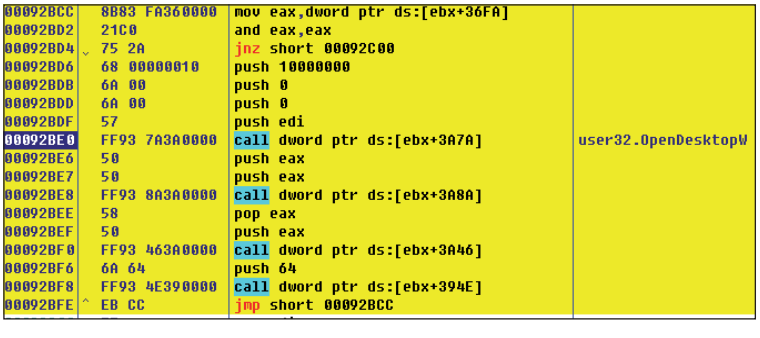


Figure 7: SwitchDesktop loop.

```
data = ((data & 0xFFFF0000) >> (key & 0xFFFF)) ^ (key
& 0xFFFF) + ((data & 0xFFFF) >> (key & 0xFFFF - 1) ^
(key & 0xFFFF - 1)
```

After decryption, it shows the following data:


00C1E4C8 osqpksxybhjhtyyqwprihcxolmfylzcqzoiivyphjwnwfmruzafrbaxjezvqgnpp
00C1E508 xxoeduegnnznuxxptwabsutuioexqzoaixzvvkzheyjvehvajpohnbteuuvtpbow
00C1E548 cgpqtfpiiqorzrnljgplhexlkouqebsyihsnfvgabfmxukakdkpaofjkbcyewggl
00C1E588 oyyjfqndxfifusouwbfpygcbeabqprkbjxkdlaxvfzqkcracnqxcoplxmvnaqtit
00C1E5C8 ptrakkftspwiltgkwfgddguypmnhxqqcdmamnosdyndixtpzdhtsrqysldcegxti
00C1E608 nmejcddfvizjrzawnfllgsbtgodapyrnqnipbipvgbbwlijzxsmycagjcnzmatst
00C1E648 sielrokffaqsrgtmhtjljuqralneblpfvrykshyhrplgdrkqfcdwvnybrtrdrcig
00C1E688 xiddcuhjcjnsoxqjmrytvluawkonmpkyjtarvpquaucomsdngtsruorkclyxkgyv

Figure 10: Decryption result.

It will use the above results and the previously prepared encrypted data to generate the C&C URL via the custom algorithm shown in Figure 11.

The result of the generation is shown in Figure 12.

In this way, it hides the basic computer information within the C&C URL, which gives the C&C's URL random characteristics and makes it hard to detect.

Response data from C&C server

After sending the information, the C&C server will send a big chunk of data back to the bot. First, let's take a look at the header – which is shown in Figure 13.

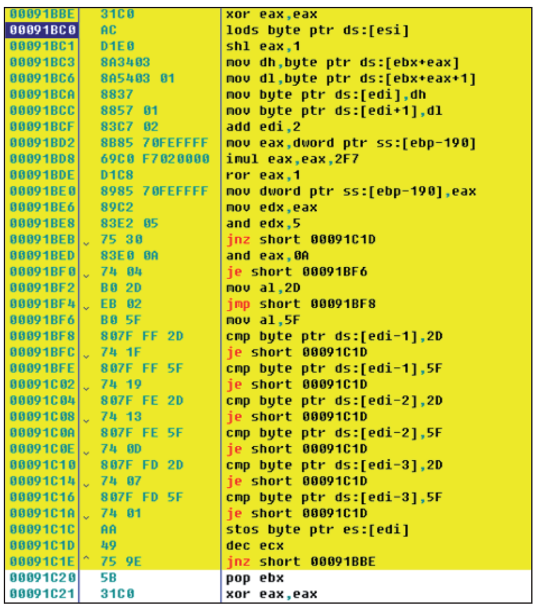

00091BBE 31C0 xor eax,eax
00091BC0 AC lods byte ptr ds:[esi]
00091BC1 D1E0 shl eax,1
00091BC3 8A3403 mov dh,byte ptr ds:[ebx+eax]
00091BC6 8A5403 01 mov dl,byte ptr ds:[ebx+eax+1]
00091BCA 8837 mov byte ptr ds:[edi],dh
00091BCC 8857 01 mov byte ptr ds:[edi+1],dl
00091BCF 83C7 02 add edi,2
00091BD2 8B85 70FEFFFF mov eax,dword ptr ss:[ebp-190]
00091BD8 69C0 F7020000 imul eax,eax,2F7
00091BDE D1C8 ror eax,1
00091BE0 8985 70FEFFFF mov dword ptr ss:[ebp-190],eax
00091BE6 89C2 mov edx,eax
00091BE8 83E2 05 and edx,5
00091BEB 75 30 jnz short 00091C1D
00091BED 83E0 0A and eax,0A
00091BF0 74 04 je short 00091BF6
00091BF2 B0 2D mov al,2D
00091BF4 EB 02 jmp short 00091BF8
00091BF6 B0 5F mov al,5F
00091BF8 807F FF 2D cmp byte ptr ds:[edi-1],2D
00091BFC 74 1F je short 00091C1D
00091BFE 807F FF 5F cmp byte ptr ds:[edi-1],5F
00091C02 74 19 je short 00091C1D
00091C04 807F FE 2D cmp byte ptr ds:[edi-2],2D
00091C08 74 13 je short 00091C1D
00091C0A 807F FE 5F cmp byte ptr ds:[edi-2],5F
00091C0E 74 0D je short 00091C1D
00091C10 807F FD 2D cmp byte ptr ds:[edi-3],2D
00091C14 74 07 je short 00091C1D
00091C16 807F FD 5F cmp byte ptr ds:[edi-3],5F
00091C1A 74 01 je short 00091C1D
00091C1C AA stos byte ptr es:[edi]
00091C1D 49 dec ecx
00091C1E 75 9E jnz short 00091BBE
00091C20 5B pop ebx
00091C21 31C0 xor eax,eax

Figure 11: C&C URL generation algorithm.

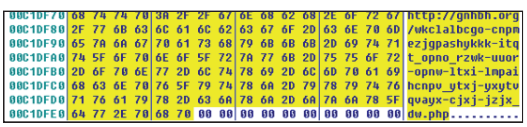

00C1DF70 68 74 74 70 3A 2F 2F 67 6E 68 62 68 2E 6F 72 67 http://gnhbh.org
00C1DF80 2F 77 6B 63 6C 61 6C 62 63 67 6F 2D 63 6E 70 6D /wkclalbcgo-cnpm
00C1DF90 65 7A 6A 67 70 61 73 68 79 6B 6B 6B 2D 69 74 71 ezjgpashykkk-itq
00C1DFA0 74 5F 6F 70 6E 6F 5F 72 7A 77 6B 2D 75 75 6F 72 t_opno_rzwk-uuor
00C1DFB0 2D 6F 70 6E 77 2D 6C 74 78 69 2D 6C 6D 70 61 69 -opnw-ltxi-lmpai
00C1DFC0 68 63 6E 70 76 5F 79 74 78 6A 2D 79 78 79 74 76 hcnpv_ytxj-yxytv
00C1DFD0 71 76 61 79 78 2D 63 6A 78 6A 2D 6A 7A 6A 78 5F qvayx-cjxj-jzjx_
00C1DFE0 64 77 2E 70 68 70 00 00 00 00 00 00 00 00 00 00 dw.php..........

Figure 12: C&C URL.

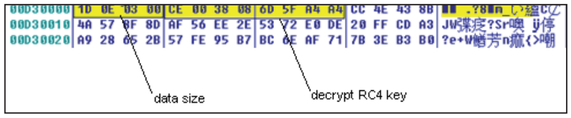


Figure 13: Header of C&C response data.

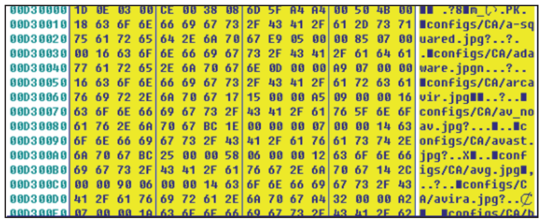

00D30000 1D 0E 03 00 CE 00 38 08 6D 5F A4 A4 00 50 4B 00 .PK.
00D30010 18 63 6F 6E 66 69 67 73 2F 43 41 2F 61 2D 73 71 configs/CA/a-sq
00D30020 75 61 72 65 64 2E 6A 70 67 E9 05 00 00 85 07 00 uared.jpg?..?.
00D30030 00 16 63 6F 6E 66 69 67 73 2F 43 41 2F 61 64 61 .configs/CA/ada
00D30040 77 61 72 65 2E 6A 70 67 6E 0D 00 00 A9 07 00 00 ware.jpgn...?..
00D30050 16 63 6F 6E 66 69 67 73 2F 43 41 2F 61 72 63 61 configs/CA/arca
00D30060 76 69 72 2E 6A 70 67 17 15 00 00 A5 09 00 00 16 vir.jpg..?..
00D30070 63 6F 6E 66 69 67 73 2F 43 41 2F 61 76 5F 6E 6F configs/CA/av_no
00D30080 61 76 2E 6A 70 67 BC 1E 00 00 00 07 00 00 14 63 av.jpg?.....c
00D30090 6F 6E 66 69 67 73 2F 43 41 2F 61 76 61 73 74 2E onfigs/CA/avast.
00D300A0 6A 70 67 BC 25 00 00 58 06 00 00 12 63 6F 6E 66 jpg?..X...conf
00D300B0 69 67 73 2F 43 41 2F 61 76 67 2E 6A 70 67 14 2C igs/CA/avg.jpg,
00D300C0 00 00 90 06 00 00 14 63 6F 6E 66 69 67 73 2F 43 ..?...configs/C
00D300D0 41 2F 61 76 69 72 61 2E 6A 70 67 A4 32 00 00 A2 A/avira.jpg?..

Figure 14: C&C response data (clear text).

After decryption with the RC4 key and data length specified in the header, it shows the data shown in Figure 14.

Note that if the DWORD (offset 0x0C) is 0x464C4553(SELF), it will quit. Only when the DWORD is 0x4b5000(PK) will it decompress the data and write it into 49 different local files which will be used to display the scam page.

In order to display the scam page, it still needs some other information, such as IP address, geo location, etc. This information is actually encrypted in another location of the response data, as shown in Figure 15.

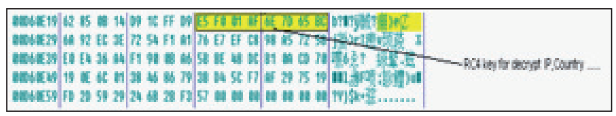


Figure 15: Additional information (key and encrypted data).

After decryption, it shows:

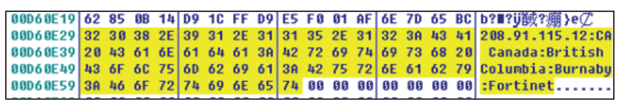

00D60E19 62 85 0B 14 D9 1C FF D9 E5 F0 01 AF 6E 7D 65 BC
00D60E29 32 30 38 2E 39 31 2E 31 31 35 2E 31 32 3A 43 41 208.91.115.12:CA
00D60E39 20 43 61 6E 61 64 61 3A 42 72 69 74 69 73 68 20 Canada:British
00D60E49 43 6F 6C 75 6D 62 69 61 3A 42 75 72 6E 61 62 79 Columbia:Burnaby
00D60E59 3A 46 6F 72 74 69 6E 65 74 00 00 00 00 00 00 00 :Fortinet.......

Figure 16: Additional information (key and encrypted data).

It will use the above information to fill the scam page and display it as shown in Figure 1.

CONCLUSIONS

LockScreen implements many different anti-debug tricks; the purpose is obvious: to make debugging tougher. However, if we are aware of those tricks and can figure out a way to bypass them, the malware will have nowhere to hide.

TUTORIAL

APKTOOL SET-UP FOR ANDROID LAB

John Foremost
Independent researcher, USA

The demand for and use of mobile devices – especially those running the *Android* operating system – are amazingly pervasive in 2013. And a wide variety of e-crime-related threats are being discovered every day for *Android*, including SMS spam, hacktivism, diallers, banking MiTM attacks, and more.

Analysis of suspect *Android* packages (APK files) starts with analysing static data, such as a cryptographic hash, and then moves onto freeware anti-virus and sandbox scans, as discussed in [1]. A researcher then typically needs to unpack the app to take a deeper look at permissions, resources, and the code. This article introduces *Apktool*, a freeware *Linux* program that is a very powerful tool for analysing APK files.

INTRODUCTION TO APKTOOL

The official home page for the *Apktool* project [2] describes it as a 'tool for reverse engineering *Android* APK files'. While it is designed for reverse engineering, using it simply to unpack and convert files can also be very useful, even for a junior analyst. One of its greatest strengths is that it enables the editing of XML in a humanly readable format, and the compiling of an edited app. The project page for this tool is well developed and documented. Its authors encourage users to join them on *Freenode* #apktool for online chatting. Their website identifies the following requirements for installation:

- JRE 1.6 (Java Runtime Environment)
- aapt command in a PATH
- basic knowledge of SDK, aapt, PATH, smali and the *Google* search engine may be useful.

That may seem a little intimidating at first, but it basically means you have to have Java and the *Android* SDK installed. Java is a platform-independent, class-based object-oriented language. 'aapt' is short for the Android Asset Package Tool, which comes with the *Android* SDK by default and is usually configured within the path variable for *Linux* accordingly. This is important so that *Apktool* can run properly when calling aapt, so that the operating system knows where to find it. This guide installs a copy of aapt in the same local directory as *Apktool* so that it can always find the file easily. The *Android* SDK is found at [3], which provides developers with tools for the building, testing and debugging of apps for *Android*.

QUICK START

Experienced users can follow the quick start steps described below to set up *Apktool* within 15 minutes or less. *Ubuntu* is the example OS for installation here:

1. Install JDK:
   sudo apt-get install openjdk-7-jre-headless
2. Download and extract apktool1.5.2.tar.bz2 and apktool-install-linux-r05-ibot.tar.bz2 from the project page.
3. Change the permissions to allow execution for each file, aapt, apktool, apktool.jar. To harden security, change to Read-Write for the owner and Read-Only for others (chmod 755).
4. Using root permissions, move the files into /usr/local/bin.
5. Type 'apktool' inside a terminal window to check functionality.

INITIAL SET-UP

I downloaded a fresh copy of ubuntu-12.04.2-desktop-i386 for this example, within a *VMware* environment. *Apktool* can be installed on *Windows*, *Linux* or *Mac* provided the requirements are met. This article reviews how to set it up in a *Linux* environment as that is my preference (as well as that of many in the security industry) for automation and analysis of *Android* code. Users of other operating systems can obtain specific instructions for their OS at the project page [2] as well as following the general guidelines provided in this article.

To get started it's a good idea to create a snapshot of your OS, if using *VMware*, so that you can return to the starting point if something goes wrong. Then start the install process by making sure Java is installed.

JRE 1.6 (Java Runtime Environment) is the first requirement. To check whether JRE is installed, enter the following command inside a terminal window: java -version. This returns the version number. If it does not exist, or you need to reinstall or update it, you can use the *Ubuntu* Software Center or apt-get to install openjdk. In a fresh installation of *Ubuntu*, Java is not installed by default, and a Java version command results in suggestions of a few packages that may be what the user is looking for. While *Apktool* requires JRE, Java Development Kit (JDK) is a more powerful package, which is aimed at developers, and is a requirement of other possibly related tools and configurations. To install JDK, enter the command 'sudo apt-get install openjdk-7-jre-headless' in the terminal.

The 'sudo' command shown in Figure 1 requires a password for elevated privileges. Once this is entered a large amount

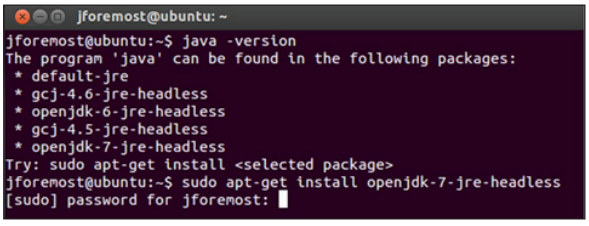


Figure 1: To install JDK, enter the command 'sudo apt-get install openjdk-7-jre-headless'.

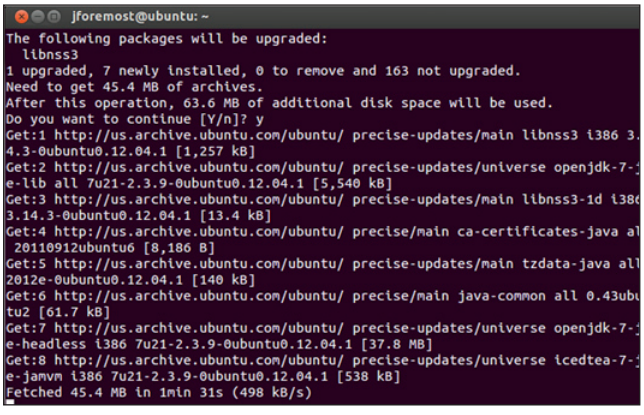


Figure 2: Following defaults to initiate part of what is downloaded for the package.

of data is pushed to the terminal. Simply answer 'Y' for yes or default settings, as prompted, to download and install the JDK package. Figure 2 shows where the user followed defaults to initiate part of what is downloaded for this package.

Naturally, a wide variety of dependencies and various updates may take place during the installation of a package like JDK. Once installation is completed, enter the 'java -version' command again to get output similar to that shown in Figure 3, confirming it is installed.

Once JDK is set up, the user can install *Apktool*. However, the project page also notes that SDK with PATH configuration for aapt is needed. On *Linux* this requires GNU C library (glibc) 2.7 or later and *Ubuntu* OS 8.04 or later (we are using 12+ in this example). While there are other granular requirements, such as 64-bit environments, a current *Ubuntu* distribution meets all the requirements to simply install SDK. A download of SDK Tools for *Ubuntu* is performed from http://dl.google.com/android/android-sdk_r21.1-linux.tgz. Once downloaded, the file is unpacked by right-clicking and selecting 'Extract Here' with the mouse or similar methods. The extracted directory should be moved to a desired location, such as the home directory for the current user. This completes the installation of SDK on the file system, although additional configuration updates

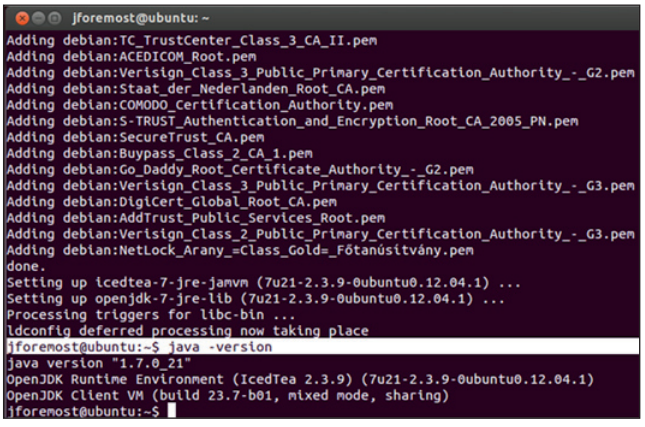


Figure 3: Output confirms installation.

may take place later through programs that use SDK (which are not covered in this article).

INSTALLING APKTOOL

The current version of *Apktool* at the time of writing this article is 1.5.2, released in February 2013. Updates take place regularly, which is great news for anyone that uses it and/or hopes to integrate it into production for analysis of code. Unlike some tools in the freeware market for *Android* security researchers, which are full of bugs and functionality issues, the current version of *Apktool* is highly robust and dependable.

Installation, no matter what OS you are using, is a matter of downloading *Apktool* and the *Apktool* install files, unpacking them, and then installing it with admin/root permissions. *Apktool* for your OS can be downloaded from http://code.google.com/p/android-apktool/. In this example apktool-install-linux-r05-ibot.tar.bz2 and apktool1.5.2.tar.bz2 are downloaded for *Ubuntu*. Extract the contents to reveal three files in total: aapt, apktool and apktool.jar.

Change permissions of the three files to read, write and execute for the owner, and read and execute only for others. (This is a permissions value of 755 for chmod geeks.) Commands like chown and chmod can be used, but in *Ubuntu* most users will simply right-click on a file, select Properties, and click on the Permissions tab. It is trivial to use this GUI method to assign permissions for each file, as desired (see Figure 4).

Once permissions are set for the files, copy or move them into /usr/local/bin. Elevated rights are needed to do this. Inside a terminal window, type 'sudo cp filename /usr/local/bin', where the filename is aapt, apktool, and apktool.jar when run for each file operation. Another easy way is to type 'sudo nautilus' inside a terminal window to open up a GUI file

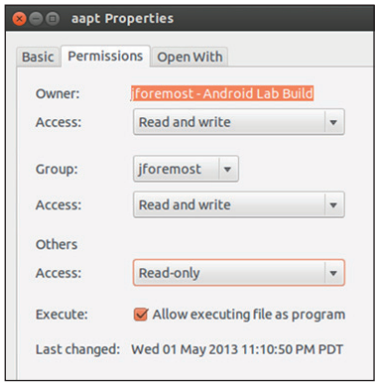


Figure 4: File permissions.

system management tool as sudo. If any error messages or dialogs appear they can probably be ignored. Simply browse to the /usr/local/bin directory and then drag and drop the three files into the *Nautilus* window. When properly installed all three files will be in the appropriate directory:

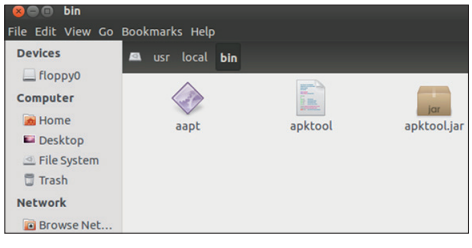


Figure 5: All three files in the appropriate directory.

To test *Apktool*, type 'apktool' in a terminal window to see the standard output (Figure 6).

It is not uncommon to try to run the tool for the first time and get an error message along the lines of '"apktool" isn't recognized as a command'. This may indicate that PATH variables are not set up correctly, that files may be missing from the two downloads required for apktool, or that the required support for JRE and SDK are not present. By following the instructions in this article a version of JDK can be installed and validated and SDK can be downloaded and placed in the user directory. Users encountering this error after following such instructions should make sure that all three files exist in the /user/local/bin directory, as shown.

USING APKTOOL

Using a terminal, simply type 'apktool' to run the tool. A common operation is to decompile an APK file to take a closer look at permissions, resources and source code. To decompile an APK file using *Apktool*, enter the following command: 'apktool d file directory'.

Figure 6: Type 'apktool' in a terminal window to see the standard output.

Figure 7 shows a terminal command for *Apktool* to decompile a file inside the user 'code' directory, with the output to be created in a new folder within that same directory called 'output':

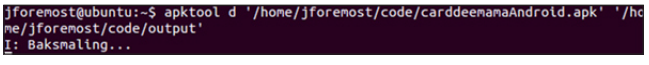


Figure 7: Terminal command to decompile a file inside the user 'code' directory.

While it can clutter a terminal window, it can be very quick simply to drag and drop a file and directory for the last two statements of a terminal command. To do this, type 'apktool d ' (including the space), and then drag and drop the APK file into the window, enter a space, and then drag and drop the output directory over the terminal window. Another shortcut is to navigate to the local directory (such as 'code' in this case), and then press the tab as you type in local filenames so that it auto-completes in whole or in part.

An efficient method is to set up a lab build where code commonly resides, such as a directory within the home directory called 'code' or 'apks'. Then create shell scripts with executable permissions simply to double-click and perform the action of the script, such as decompiling all files found within the code directory.

Output of a decompiled APK using *Apktool* includes a yml file, AndroidManifest.xml, smali, assets, and res directories. Humanly readable decoded XML is a fantastic feature of APKs decoded by *Apktool*. Additionally, they can be edited and then recompiled/packaged using *Apktool*, which is also very powerful for analysis and debugging research. An image of a decompiled XML file is shown in Figure 8.

A review of the humanly readable XML manifest file for the app analysed reveals two important permissions related to boot and the SD Card. Moghava is an *Android* threat that 'stamps' images on the SD card with a political/religious figure. *Apktool* makes it easy to decompile and quickly

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<manifest android:versionCode="2" android:versionName="1.1" package="ir.sharif.iranianfoods">
 <application android:label="@string/app_name" android:icon="@drawable/icon">
  <activity android:theme="@*android:style/Theme.NoTitleBar.Fullscreen" android:label="@string/app_name"
   android:name="StartActivity">
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
  </activity>
  <activity android:label="@string/app_name" android:name=".TabHostActivity"
  android:screenOrientation="portrait"/>
  <activity android:label="@string/app_name" android:name=".Ostans"/>
  <activity android:label="attractions" android:name=".Attractions"/>
  <activity android:label="information" android:name=".Information"/>
  <activity android:label="touch" android:name=".Touch"/>
  <activity android:label="favorites" android:name=".Favorites"/>
  <activity android:label="search" android:name=".Search"/>
 <service android:name="com.Moghava.stamper">
  <intent-filter>
    <action android:name=".stamper"/>
  </intent-filter>
 </service>
 <receiver android:name="com.Moghava.kicker">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
  </intent-filter>
 </receiver>
</application>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
</manifest>
```

Figure 8: Decompiled XML file.

inspect the manifest XML for such permissions and related actions as an analyst begins analysis of a possible APK threat.

Apktool has many other powerful options. One of the most obvious is the 'b' option, to build an app from the modified code. This enables a researcher or developer to decompile, modify, and recompile an app which lends itself to a multitude of possible applications. Other parameters exist, such as 's' where only the resources are decompiled and the source code is not, to speed up operations if a developer only desires to make a few changes to resources before recompiling an app.

Apktool also makes it possible to debug smali source code step by step. This is true debugger operational capability which reveals more advanced and powerful features of *Apktool*. For more information on Smali debugging using *Apktool* go to [4].

Apktool is a very powerful tool for security researchers concerned with analysing *Android* threats. Using the steps described in this article, researchers should be able to set it up and start making use of its many useful features.

REFERENCES

[1] https://www.virusbtn.com/virusbulletin/archive/2012/02/vb201202-mobile-malware-analysis.

[2] http://code.google.com/p/android-apktool/.

[3] http://developer.android.com/sdk/index.html.

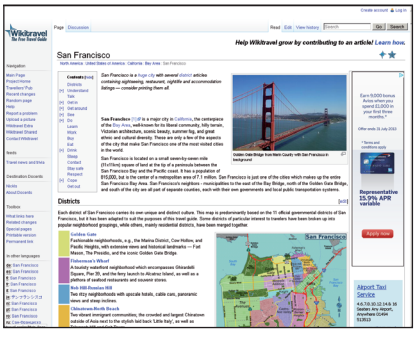
[4] http://code.google.com/p/android-apktool/wiki/SmaliDebugging.

SPOTLIGHT

GREETZ FROM ACADEME: ETHICAL QUANDARIES

John Aycock
University of Calgary, Canada

There is often a disconnect between academic security research and anti-malware industry research – in both directions. In the 'Greetz from Academe' series, Dr John Aycock, Associate Professor at the Department of Computer Science, University of Calgary, picks out some of the work going on in academic circles and summarizes the key points – Ed.



This month, I'd like to tell you about a trip I took to San Francisco. San Francisco is a beautiful city, with world-famous sights such as Fisherman's Wharf, Alcatraz, and the Golden Gate Bridge. Or so *Wikitravel* tells me [1] – I was there for a workshop and saw exactly none of it.

As for why this particular workshop was interesting, I should perhaps back up a bit. I have always found the AV community to be very sensitive to ethics, and concerned with acting ethically. Sometimes it's just statements in conversation of the form '*X* is unethical', and at other times it's on a larger scale. For example, I recall one *VB* conference presentation some years back in which the presenters had accessed a botnet's C&C server and had debated using the botnet's 'kill' command to try and clean the compromised computers in the botnet. It was remarkable to witness the audience's wide eyes and the collective drawing in of breath as the ethical and legal ramifications of that action hit home. Even the vetting process for new recruits used by one AV company sounded like an evaluation not just of technical chops, but of whether or not the potential employee shared the same ethics as the team.

Ethics figures prominently in academic security research, too. Of course, ethics itself is ancient, dating back to Aristotle harping on about virtue and wondering how to keep his toga from chafing, but there are modern flavours that are more specialized – like computer ethics [2, 3], although they pay scant attention to the toga issue. The professional organizations that many academics belong

to, such as the ACM and IEEE, also have codes of ethics [4, 5].

Additionally, there is quite a bit of research oversight. For me to undertake academic research with humans, or gather data from humans, I first have to submit an ethics application to my friendly neighbourhood research ethics board (sometimes called an institutional review board, or IRB). In it, I have to detail everything I want permission to do, which includes not just the research itself, but also how I'm planning to recruit people, how they'll give informed consent, and how I'll be storing and disposing of data. All of this is mandatory, courtesy of the atrocities that occurred during World War II and sundry dubious experiments in the last century.

However, if the emperor can be said to have no clothes, then the emperor of academic security research ethics... well, let's just say that he can sometimes count to 21 without difficulty. Academics don't have to belong to professional organizations, and it's not clear how strictly those codes of ethics are enforced anyway or, as the ACM Code itself [4] describes the worst case: 'membership in ACM may be terminated'. Ouch.

Review by a research ethics board works well for research involving humans, because that's where the research guidelines they follow derive from. Throwing computers into the mix doesn't help [6]. Being a computer security expert is not a prerequisite for reviewing ethics applications; subtle but critical nuances may be missed. There are also plenty of loopholes for security research. Say that I'm building an undetectable, highly destructive piece of malware in my lab. (Chill out, I'm not.) I'm not doing research with humans, and therefore my work requires no ethics oversight, even though it should probably have some: if the dreaded W32/Aycock escapes my lab, it may have a profound impact on humans.

This extends to academic publication venues. Let me pick on WOOT, for instance, the USENIX Workshop on Offensive Technologies [7]. As the name implies, the workshop is all about new attack methods. Of the 60 papers that have appeared there between 2007 and 2012, how many have contained any mention of ethics? Two. While heated discussion of ethics may be taking place behind closed doors, very little heat seems to leak out.

Change is coming, starting at the academic grassroots level. The workshop I attended this May in San Francisco was CREDS, the Cyber-security Research Ethics Dialog & Strategy workshop [8], and this was not the first such event. Three workshops known as WECSR, the Workshop on Ethics in Computer Security Research, were run between 2010 and 2012 [9]. There is also a recent set of guidelines on how ethical security research may be conducted, called the Menlo Report [10].

Academic publication venues are changing too. The calls for papers for some notable security venues, such as SOUPS (usable security), PETS (privacy), and the USENIX Security Symposium [11–13], now include an ethics requirement.

In conclusion, some flawed processes aside, ethics are a common concern in the AV community as well as in academic security research. Prevention of toga chafing, on the other hand, is still an open question.

REFERENCES

[1] Wikitravel. http://wikitravel.org/en/San_Francisco.

[2] Baase, S. A Gift of Fire: Social, Legal, and Ethical Issues for Computing Technology, 4th edition. Prentice Hall, 2012.

[3] Johnson. D.G. Computer Ethics, 3rd edition. Prentice Hall, 2001.

[4] ACM Code of Ethics and Professional Conduct. http://www.acm.org/about/code-of-ethics.

[5] IEEE Code of Ethics. http://www.ieee.org/about/corporate/governance/p7-8.html.

[6] Buchanan, E.; Aycock, J.; Dexter, S.; Dittrich, D.; Hvizdak, E. Computer Science Security Research and Human Subjects: Emerging Considerations for Research Ethics Boards. Journal of Empirical Research on Human Research Ethics 6(2), 2011, pp.71-83.

[7] USENIX Workshop on Offensive Technologies. https://www.usenix.org/conference/woot13.

[8] Cyber-security Research Ethics Dialog & Strategy Workshop (CREDS). http://www.caida.org/workshops/creds/1305/.

[9] Workshop on Ethics in Computer Security Research (WECSR) http://www.cs.stevens.edu/~spock/wecsr2012/ (contains links to the previous years of WECSR workshops).

[10] Bailey, M.; Dittrich, D.; Kenneally, E.; Maughan, D. The Menlo Report. IEEE Security & Privacy, March/April 2012, pp.71-75.

[11] Symposium On Usable Privacy and Security. http://cups.cs.cmu.edu/soups/2013/cfp.html.

[12] Privacy Enhancing Technologies Symposium. http://petsymposium.org/2013/cfp.php.

[13] USENIX Security Symposium. https://www.usenix.org/conference/usenixsecurity13/submitting-papers.

END NOTES & NEWS

TakeDownCon Rocket City takes place 11–16 July 2013 in Huntsville, AL, USA. 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 details 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 8th Annual (ISC)² SecureAsia takes place 7–8 August 2013 in Manila, Philippines. See http://www.informationsecurityasia.com/.

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

ZebraCon 2013 takes place 27–29 August 2013 in Kuala Lumpur, Malaysia. For details see http://zebra-con.com/home/.

Cyber Intelligence Europe takes place 17–19 September 2013 in Brussels, Belgium. For details see http://www.intelligence-sec.com/events/cyber-intelligence-europe.

Hacker Halted USA will take place 19–21 September 2013 in Atlanta, Georgia, USA. For more information see https://www.hackerhalted.com/2013/us/.

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

SecTor 2013 takes place 7–9 October 2013 in Toronto, Canada. For details see http://www.sector.ca/.

ISSE 2013 will take place 22–23 October 2013 in Brussels, Belgium. For more details see http://www.isse.eu.com/.

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/.

RSA Conference Europe takes place 29–31 October 2013 in the Netherlands. For details see http://www.rsaconference.com/events/2013/europe/index.htm.

The First Workshop on Anti-malware Testing Research (WATeR 2013) takes place on 30 October 2013 in Montreal, Canada. For full details see http://secsi.polymtl.ca/water2013/.

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.

AVAR 2013 will take place 4–6 December 2013 in Chennai, India. For details see http://www.aavar.org/avar2013/.

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.

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/