JULY 2008

virus BULLETIN

Fighting malware and spam

CONTENTS

ISSN 1749-7027

IN THIS ISSUE

MEMORY GAME

Eric Filiol describes a set of computer memory weaknesses that could enable the theft of sensitive data via malware attacks.
page 6

SNAKES AND LADDERS

The object-oriented programming language Python can be used for many kinds of software development – potentially including malware development. Aleksander Czarnowski believes in being prepared and provides a brief overview of how to reverse engineer a Python module.
page 10

THE WAITING GAME

This month John Hawes got his hands on a beta version of the long-awaited *VIPRE* from anti-spyware expert *Sunbelt Software* and found it to be well worth the wait.
page 16

vbSpam supplement

This month: anti-spam news and events, and Paul Baccas questions whether spear phishing is on the rise.

'The purpose of the VB100 is to provide a regular measure of the competence, reliability and credibility of software vendors in the security field.'

John Hawes, Virus Bulletin

A COMMITMENT TO QUALITY AND RELIABILITY

The VB100 certification system has come under fire in recent weeks, with much of the criticism focused on the WildList and its suitability as a basis for testing. It became quite clear from the stories that were published that there are several common misconceptions surrounding both the intended purpose of the VB100 certification, and in particular the WildList.

One of the central criticisms levelled at the WildList is that it does not include every piece of malware. To do so, of course, would be an immeasurably huge task beyond even the vast resources of large globe-straddling corporations. It would also be quite beside the point of both the WildList and the certification schemes that rely on its steady and regular output.

There have been numerous other criticisms of the WildList, most of which focus on the range of malware types covered by the list and the activeness of its reporting sources. These are issues into which the team behind the WildList are investing considerable effort to address. But even once the full range of improvements are fully on stream, the WildList will never pretend to cover the gamut of malicious software; rather it is intended to provide a limited, but unquestionable subset of the malware problem, containing items which are guaranteed to be affecting a significant proportion of real-world users and represented by a set of rigorously validated master samples.

Tests that pit products against the WildList have never claimed to prove that a given product can detect all known malware (which would be impossible to prove) and they do not attempt to rank products against one another on the basis of detecting more or fewer of the samples listed. The purpose of the VB100 and similar certification schemes is to provide a regular measure of the competence, reliability and credibility of software vendors in the security field – something which has become more important than ever in recent years with the growing tide of suspect software claiming to detect and remove malware.

Products are expected to be able to pass *VB*'s tests, and to pass regularly. With the level of co-operation and sample sharing going on across the industry, nothing on the list should be new to vendors, and with the comparatively tiny resources of the *VB* test lab in relation to the extensive research labs that AV vendors have at their disposal, no amount of replication of complex viruses carried out by *VB* should be beyond the capabilities of a commercial malware lab.

Passing, or even failing a single VB100 test means little in isolation – it is all about maintaining a steady record of passes over time, to demonstrate a long-term commitment to quality and reliability.

Of course, beyond these issues, there are far more complex and difficult problems facing testers. An ever-growing arsenal of weapons is being implemented in a diverse range of fashions as products adapt to combat the evolving threat. Testing these new weapons – and just as importantly interpreting and presenting the results in a manner comprehensible to the end-user – is a hard but vital task, and one that *VB*, like all testing bodies, is facing up to. We are hard at work developing a range of improvements and additions to the data we provide to our readers, and are currently hiring extra hands to cope with the requirements of testing a wider range of criteria and maintaining a broader and more up-to-the-minute sample collection.

For any such plan to work requires the input and co-operation of experts from across the industry, pooling both wisdom and resources for the greater good. Groups such as AMTSO provide great hope for the future, and a number of the presentations at this year's *VB* conference will focus on the subject of testing. As we strive to provide useful and trustworthy data on the protection offered by a growing range of solutions to the security problem, we rely on the support of those whose performance we measure, as they rely on independent tests to keep them informed of their successes and failings. As always, we gladly welcome new ideas and constructive criticism.

NEWS

IT HEAVYWEIGHTS COMBINE FORCES TO FIGHT CYBER CRIME

A new security industry consortium was formed last month to provide a forum for IT vendor companies to work together in order to address multi-vendor security threats. The Industry Consortium for Advancement of Security on the Internet (ICASI) is a collaboration between *Cisco*, *International Business Machines*, *Intel*, *Juniper Networks* and *Microsoft,* and is organised around four central principles: reducing security threat impact and improving customer security; improving the efficiency and effectiveness of multi-vendor threat resolution and security response practices; creating a unique trusted environment for the sharing of information between vendors; and leveraging the expertise of IT companies from across the world to innovate security response excellence. The organization plans to share its first accomplishments in late 2008.

LIAR, LIAR

CEO of *Trend Micro* Eva Chen surprised many last month when she stated in an interview with ZDNet that the entire anti-virus industry has been lying to its customers for the past 20 years.

Trend has recently announced that it is heading in a new direction (into the cloud) with its malware analysis – reasoning that, now that faster Internet connections are available worldwide, it is faster to throw an unknown sample into the cloud to perform a suspected malware check than to initiate and execute a sandbox heuristic environment on the desktop.

FAST FLUX TROJAN AUTHOR IN COURT

A 19-year-old is due to plead guilty in a US court to one count of computer assisted fraud after having admitted to creating the Nugache trojan and using it to create one of the first fast flux botnets. The trojan spread through *AOL* instant messenger and, once clicked on, added the victim machine to a zombie network that used a peer-to-peer mechanism to communicate rather than relying on a single command and control channel.

According to a plea bargain agreement Jason Michael Milmont ran a botnet using the trojan which, at its peak, consisted of between 5,000 and 15,000 computers. He used the botnet to obtain victims' credit card details and steal thousands of dollars by making online purchases using the stolen credentials. The botnet was also used to launch DDoS attacks against an online business.

Milmont faces up to five years in prison, a $250,000 fine and almost $74,000 restitution.

Prevalence Table – May 2008

Malware	Type	%
NetSky	Worm	25.20%
Agent	Trojan	15.68%
Rays/Traxg/Wukill	Worm	10.12%
Mytob	Worm	9.38%
OnlineGames	Trojan	8.03%
Cutwail/Pandex/Pushdo	Trojan	7.01%
Virut	Virus	4.34%
Bifrose/Pakes	Trojan	3.99%
Mydoom	Worm	3.46%
Bagle	Worm	2.69%
Zafi	Worm	1.89%
Zlob/Tibs	Trojan	1.40%
Grew	Worm	1.02%
Sality	Virus	0.84%
Mywife/Nyxem	Worm	0.65%
Stration/Warezov	Worm	0.39%
Nuwar/Peacomm/Zhelatin	Trojan	0.37%
Bugbear	Worm	0.34%
Lineage/Magania	Trojan	0.29%
Small	Trojan	0.29%
Alman	Worm	0.27%
Feebs	Worm	0.22%
Klez	Worm	0.21%
MyLife	Worm	0.18%
Chir	Worm	0.14%
Parite	Worm	0.14%
Grum	Worm	0.13%
Bagz	Worm	0.13%
Nimda	Worm	0.12%
FunLove	Worm	0.12%
Womble	Worm	0.11%
Vote	Worm	0.08%
Delf	Trojan	0.07%
Others[1]		0.71%
Total		100.00%

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

VIRUS ANALYSIS

THE ROAD LESS TRUVELLED: W32/TRUVEL

Peter Ferrie
Microsoft, USA

Everything old is new again – at least for some virus writers.

By the addition of a relocation table, *Vista* executables can be configured to use a dynamic image base. That essentially turns them into executable DLLs. Now a virus has come along that has made a 'breakthrough' by infecting these executables – at least it would be a breakthrough if it weren't for the fact that relocatable executables have been supported since *Windows 2000* (ASLR in 1999!), and we have seen plenty of viruses that can infect DLLs. What's more, applications can have different image bases even without a relocation table, which from the virus's point of view amounts to the same thing. There is no need for a virus to carry absolute addresses – the alternative is a technique called 'relative addressing'.

LOCK AND LOAD

The virus, which we call W32/Truvel, begins by saving all registers and flags using the 'pusha' and 'pushf' instructions, as well as saving the ebp register explicitly (perhaps the virus author thought that the 'pusha' instruction was not sufficient). Then the virus determines its load address. This can be done simply by using a call→pop sequence, but the virus author seems to have wanted to make it more complicated. In this case, the load address is determined by calling a routine that sets up a structured exception handler, then intentionally causes an exception.

The handler receives control and retrieves the pointer to the context structure. It retrieves the original esp register value from the context structure, then fetches the return address from the stack and uses it to calculate the delta offset. The offset is stored in the ebp register within the context structure. Then the handler adjusts the eip register value in the context structure in order to skip the instruction that caused the exception, and returns control to the operating system to resume execution.

Interestingly, the handler contains an instruction to retrieve the base address of the Process Environment Block, but does nothing further with it. It is unclear what purpose this might have served in an exception handler. The first version of the virus also contains a check for the presence of a debugger by examining the 'BeingDebugged' flag in the Process Environment Block, but there is no branch instruction to take action if the flag is set – perhaps it was removed while debugging, and the virus author forgot to restore it. In the second variant of the virus the sequence has been removed completely.

SUCH HOSTILITY

Upon returning from the exception handler, the virus checks for the presence of a debugger by examining the 'BeingDebugged' flag in the Process Environment Block. If a debugger is detected, then the virus branches intentionally to an invalid address (which is the value of the eflags register), and the process terminates.

CRASH AND BURN

If no debugger is detected, the virus saves two image base values on the stack: the image base value from the Process Environment Block and the kernel32.dll image base value which it retrieves from the InLoadOrderModuleList structure. This can lead to a problem, but only in the most unlikely circumstances, such as a bad memory layout in an emulator. Part of the problem is that if the kernel32.dll image base does not contain the right signatures (i.e. beginning with 'MZ', and with the lfanew field pointing to the PE header), then the virus attempts to clean up and run the host. The other part of the problem is that, at that point, no API addresses have been retrieved, so the cleanup will probably cause the application to crash.

In case the addresses saved during replication happen to match, the virus attempts to free two memory blocks that it has not yet allocated. This may not cause a crash, but another problem is caused by the fact that the two image base values that were saved onto the stack are not removed prior to the virus attempting to restore the registers and flags – which results in register corruption. However, even that might not be enough to cause a crash. The fatal blow comes in the form of the host entrypoint not having been adjusted according to the image base value, so the virus always branches to an invalid memory address and crashes.

Another bug exists in the code that attempts to locate the GetProcAddress() API. The virus loops through all of the APIs until GetProcAddress() is found. However, if for some reason the function is not found and the loop exits, the code continues its execution at the same location as that which is reached if the function is found. The result is that the virus resolves to an address which will likely point to an invalid memory address and cause a crash.

PROTECT AND SERVE

The virus calls VirtualProtect() to write-enable its code. This is the result of an anti-heuristic effect which will be explained below. If the call to VirtualProtect() fails for some reason, then, as above, the virus branches to the cleanup routine and crashes.

At this point, the virus removes the image base values from the stack, and adjusts the host entrypoint according to the image base value. Then comes some code of great silliness:

The virus wants to retrieve the addresses of some functions from kernel32.dll. While it is a simple matter to construct one relative pointer to the list of names and one relative pointer to the location at which to store the addresses, the virus writer chose another method. The virus carries a table of pairs of absolute addresses. One half of the pair points within the virus code to the name of the function to retrieve from kernel32.dll, while the other half points within the virus code to the location at which to store the retrieved address. Each of the addresses must be adjusted individually according to the delta offset, in order to locate the appropriate data. If any of the functions cannot be resolved, then the virus branches to the cleanup routine and attempts to free two memory blocks that it has not yet allocated. The list of functions includes entries that the virus does not even use.

LOSING MY MEMORY

The virus calls a function twice to allocate two blocks of memory for itself. However, after each call comes a check for failure. If the first allocation fails, then the virus branches to the cleanup routine and attempts to free the second block which it has not yet allocated.

If the allocations succeed, then the virus searches in the current directory for all files whose suffix is '.exe'. For each file that is found, the virus opens it and reads some data into one of the memory blocks. The virus checks for the 'MZ' signature, and the second variant includes some bounds checking on the lfanew field value prior to checking for the PE signature. The problem is that the bounds checking is incorrect.

Instead of checking whether the lfanew field value plus the size of the signature is not greater than the size of the block, the virus attempts to check only if the lfanew field value is less than the size of the block – and it even gets that wrong. The virus checks that the lfanew field value is not greater than the size of the block. This allows for an lfanew value that is exactly equal to the size of the block – also known as an off-by-one bug.

The problem is compounded by the fact that no further bounds checking is performed, leading to the assumption that if the PE header signature is within the block, then the entire PE header and the section table must be within the block.

EVUL IS AS EVUL DOES

The infection marker for the virus is a section named 'Evul', which is the name of the virus author. If no such section exists, then the virus simply appends one, without regard to the possible overflow of the block or the overwriting of the data in the first section. The virus then seeks the end of the file and calculates a new size according to the FileAlignment field value. If the file size was not aligned before, then the virus attempts to write enough data to align it. However, the stack is the source of the data to write, and if the amount of data to write is large enough, then it will fail. This result is not checked.

The virus calculates an aligned SizeOfRawData value for the original last section. If the value was not aligned already, then the virus replaces the old value with the new one, and applies the difference to the SizeOfImage value. This is another bug, since the SizeOfImage value comes from the sum of the VirtualSize values, not the sum of the SizeOfRawData values.

BACK AND FILL

The rest of the data for the new section are filled in at this point. The virtual address is calculated by aligning the VirtualSize of the previous section. The section characteristics specify a section that is read-only and executable. In the past, it was common for viruses to make the last section writable when they infected a file. It became such a common technique that some anti-virus programs still use it as a rule for performing more thorough scans of files. As a result, the absence of the writable bit can help some viruses to hide, at least for a while.

Next, the virus zeroes the LoadConfig and BoundImport data directory entries in the PE header. This has the effect of disabling the Safe Exception Handling, since the entries are located inside the LoadConfig data.

Finally, the virus writes itself to the file, updates the entrypoint to point to the new section, and writes the new PE header to the file. Then the virus searches for another file to infect.

The virus has no intentional payload, however its many bugs are sufficient to produce some surprises – it's amazing that the virus replicates at all.

FEATURE 1

NEW MEMORY PERSISTENCE THREATS

Eric Filiol
ESAT, France

Recent research has shown that, contrary to popular belief, the content of computer memories (RAM) is not erased when a computer is shut down. Different kinds of data can survive even after events that should normally result in the RAM being erased and reset to zero: program termination, shutdown or switching off the computer. The survival of data in RAM may not only affect the security of cryptographic applications but may also be used efficiently to design new, powerful malware threats.

This article first presents an in-depth analysis of computer memory weaknesses that could enable the theft of sensitive data via malware attacks. Most of these attacks are made possible due to the persistence properties of modern computer memory modules. In the second part of the article, we present different attack methods that have been identified and tested and which could be used maliciously. In the final part we present some tools and security policy enhancements that should greatly contribute to preventing or limiting those attacks.

THE PROBLEM: PERSISTENCE OF COMPUTER MEMORY MODULES (RAM)

State of the art: memory remanence

For a long time it was widely believed that computer memory modules (aka Random Access Memory or RAM) were erased (reset to zero) immediately after a program terminates or a computer is shut down, thus causing their content to disappear from the computer. However, a number of studies have shown this assumption to be partially wrong. A number of studies [1–3] have identified risks attached to what is known as memory remanence:

> '...Ordinary DRAMs typically lose their contents gradually over a period of seconds, even at standard operating temperatures and even if the chips are removed from the motherboard, and data will persist for minutes or even hours if the chips are kept at low temperatures. Residual data can be recovered using simple, non destructive techniques that require only momentary physical access to the machine…' [3]

The authors of [3] observed a data remanence effect at normal operating temperatures (between 25.5 °C and 41.1 °C) after 2.5 to 35 seconds (depending on the computer) with a binary error rate ranging from 41% to 50%. They managed to increase this remanence time to 60 seconds with a very negligible error rate, simply by cooling the RAM at a temperature of -50 °C.

From those results, the researchers identified a number of security risks with respect to data remanence. In particular, they explained how secret cryptographic keys could illegitimately be retrieved by exploiting the RAM remanence property. Despite the undisputed interest of this study, its operational scope is rather limited: the attacker must have physical access *and* must cool the RAM immediately after a sensitive application has been executed (e.g. encryption/decryption). Except in the case of investigation by police forces, this attack remains of theoretical interest only.

At the time of publication of [3], another team was working on the same subject but with a broader, more operational approach and at normal operating temperatures by considering the concept of RAM persistence [4].

Memory data persistence

RAM data remanence considers only the physical, electronic effects that enable data to survive temporarily in RAM. But data disappearing from memory does not necessarily mean that the data has disappeared from the computer, and in many cases, memory contents remain available inside the computer for a very long time: we call this *memory data persistence*. Let us adopt the following definition:

Memory data persistence [4]: the set of both physical (remanence) and operating system effects/mechanisms that cause data to survive in RAM and/or in a computer after a program terminates or a computer is shut down.

Without entering into too much detail, besides the single remanence effects that have been confirmed and developed further, we have identified a number of other mechanisms which preserve the content of memory modules. The main ones can be summarized as follows:

- Swap files (the pagefile.sys file under *Windows*, and the swap partition under *Linux*) generally contain all or part of the memory.
- Hibernation files (the hiberfil.sys file under *Windows*) contain a lot of memory data.

Data persistence mechanisms	Security risk
RAM remanence	1
Swap file	3 – 4
Hibernation file	2
Memory dump file	3 – 4

Table 1: Security risk with respect to data persistence (lowest = 0 highest = 4).

- The *Windows* memory dump file (MEMORY.dmp) contains the whole RAM content.

Table 1 summarizes the level of computer security risk attached to those mechanisms.

The different experimental results (see [4] for details) clearly demonstrate that the operating system saves the RAM content very frequently (wholly or partly) into dedicated files, thus causing critical data to survive far longer than expected, even after the computer has been rebooted. All of those mechanisms, besides the RAM remanence effect, constitute a critical risk against data confidentiality.

MALWARE-BASED ATTACKS AGAINST CONFIDENTIALITY

Let us first consider how a dedicated piece of malware could exploit the RAM persistence in a computer. In other words, we consider the different actions that a piece of malware could take to steal critical data with respect to memory persistence. We will present only a few of the most illustrative examples included in [4]. We will consider a piece of malware that is undetected by anti-virus products as a general framework.

Eavesdropping confidential data

In this case, the malware will look for sensitive data that survive either in memory (remanence) or in memory dump files. It is worth mentioning that the malware itself may induce the creation of such files.

- Let us suppose that a secret (inert or not) file is processed (scanned by an anti-virus engine or processed by a dedicated application) on a computer that is infected with a piece of malware. A %SystemRoot%\MEMORY.DMP file is created. In most cases, this file will contain at least a significant part of the secret data. In some cases, it is possible to steal plain-text data during the decryption of an encrypted document.
- A piece of malware can explore the computer's RAM content directly in order to find secret data. Even after a few hours, in some cases, the information remains in memory. As an example, we plugged in a USB key containing a secret file and then unplugged it. It was still possible after the USB key had been removed to find a lot of data with respect to the file (the experiment can be reproduced by using the *WinHex* software which embeds a forensics function called '*RAM editing*').
- Secret data is also saved by *Windows XP* in the hibernation file HIBERFIL.SYS. Any piece of malware could very easily access this file and retrieve a lot of data that is contained in RAM when the computer goes into sleep mode. If sleep mode is not activated by default, the malware is able to activate it.

There are also many more ways for malware to collect sensitive data – even when it is protected by encryption – by exploiting the data persistence.

Theft of password or encryption keys

Now let us see how a piece of malware could collect critical data with respect to the security of the computer itself: password and encryption keys.

Analysis of the *Windows* swap file (PAGEFILE.SYS) or of the hibernation file may reveal such critical data, as well as it appearing in the %SystemRoot%\MEMORY.DMP file. As an example, let us consider the PAGEFILE.SYS file. The

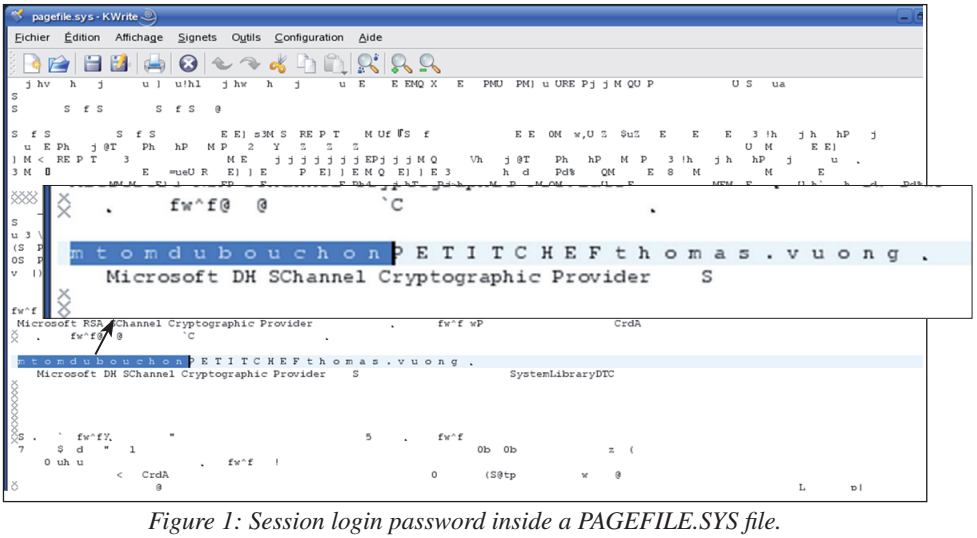


Figure 1: Session login password inside a PAGEFILE.SYS file.

session password can survive totally or partially in that file, even after a reboot. The most interesting thing is that we can recover the passwords of different users (in multi-user mode). Figure 1 shows the presence of a session login password inside the PAGEFILE.SYS file (nine characters out of a total of 11 are recovered).

As for encryption keys, the data persistence (including data remanence) will depend partly on the security enforced at the application level. Tests have been conducted on different pieces of encryption software. For some of them, it is possible to retrieve wholly or partly the password used to protect private keys in public key encryption applications, and even the private key itself can be retrieved either in the HIBERFIL.SYS file or in the MEMORY.DMP file. In some cases, the private key may also be present inside the PAGEFILE.SYS file. Figure 2 shows the presence of the encryption password inside a *Windows* hibernation file after decryption with the open-source *Cryptonit* [5] software. (The same applies with various other encryption software packages.)

It is therefore essential to keep in mind that the security provided by the operating systems (some of the same results have been obtained under *Linux*) and/or the security software (e.g. encryption application) is not watertight and critical data such as passwords and encryption keys may be leaked. Even if such data is only partly recovered by a malicious attacker (most of the time the recovery rate is higher than 80%) it will be easy to guess the remaining part (e.g. using a reduced brute force approach).

The other essential point lies in the fact that most of the system files we have considered (swap file, hibernation file, memory dump file) can be created by any malware itself. It only has to manipulate the appropriate system configuration files and access the appropriate system description tables (e.g. ACPI tables).

NEW MALWARE CONCEPTS EXPLOITING MEMORY PERSISTENCE

In this section we will explain how data persistence can be exploited by a piece of malware to replicate (self-reproducing codes) or just to operate (installation of simple malware such as trojans, logic bombs etc.). The payload will not be taken into account here. We will present a very simple, yet powerful proof of concept.

Before revealing the general mechanisms operated by our proof of concept, it is essential to make one very important point clear. For the attacker, the main problems with data persistence (especially the remanence part) lie in the fact that the data can only partly be recovered and in the fact that he does not know *a priori* what that data is. For example, in the case of obtaining a secret key, the exact location of the key and the amount of remaining information that needs to be guessed following data recovery may make the attack more complex than expected [3]. Moreover, the attacker cannot initially operate on the data that is supposed to be persistent in memory (using error correcting techniques for example).

But in the context of a piece of malware that is going to exploit data persistence to operate, the initial preparation of data used for that purpose is possible. In other words, the malware will always know what it is looking for and where to find it. We just have to use error correcting techniques to prevent data loss due to the natural and random limitation of data persistence (including data remanence).

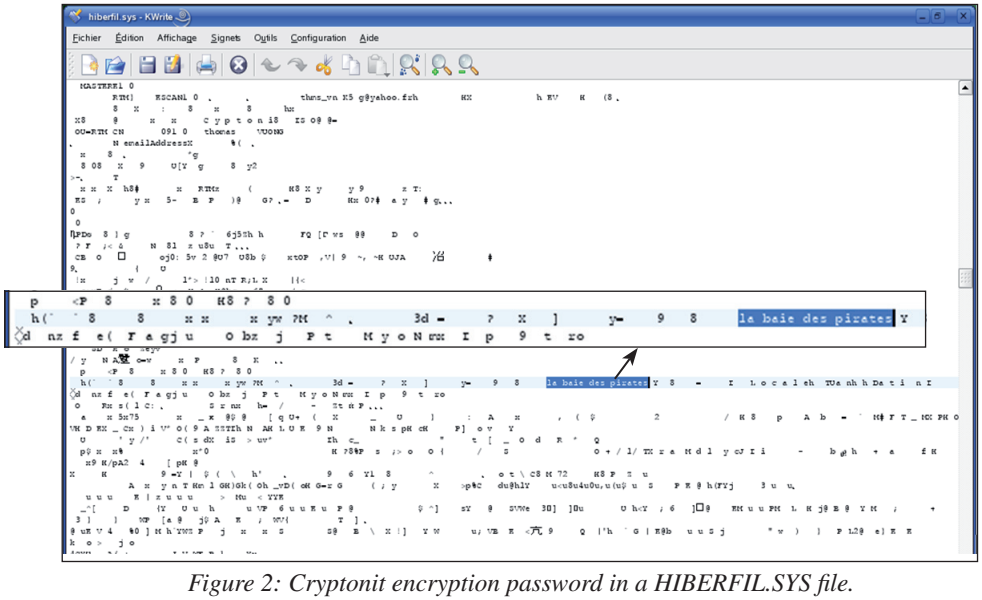


Figure 2: Cryptonit encryption password in a HIBERFIL.SYS file.

The general design of the proof of concept combines the data persistence effects with the most sophisticated malware techniques that have recently been identified:

- *K*-ary codes [7, 9]. Instead of having a single file containing all the malicious information, *k*-ary malware are composed of *k* different files, each of which looks innocuous. A suitable combination – either serially or in parallel – of (at least) a subset of those *k* parts results

in the malware operation. In [9], it was shown that detecting *k*-ary malware is an intractable problem. One very interesting approach for malware is both to split the malicious information and to introduce a time delay between them. In this respect, data persistence can provide a very powerful set of techniques to realize such codes. A significant subset of those *k* parts can simply be persistent data either in memory (remanence) or in some system files.

- Strongly armoured codes [6–8]. Such malware is encrypted with strong algorithms (e.g. AES, RC6, and Serpent), but unlike most encrypted malware the secret key is not stored inside the code. In this setting, the key is only available as a quantity taken from the environment and is basically under the attacker's control. In our context, this key may be taken from data that is known to be persistent in the computer at a given time or after a given event, under the attacker's control.
- Cryptography-based obfuscation techniques [7, 10]. This approach is quite similar to the previous one, however in this case it is not the key which is the information taken from data known to be persistent in the computer at a given time/after a given event, but the obfuscation algorithms themselves.

All these techniques have been tested and have proven to be very efficient. This shows that data persistence represents extraordinary potential for developing existing malicious techniques further, in a very sophisticated way.

PREVENTION

The essential question is: how can we prevent or limit the exploitation of data persistence by malware, since detecting such sophisticated code is bound to be a very complex challenge.

The following methods should greatly contribute to preventing such attacks:

- Anti-virus software should scan the entire memory systematically and not only the memory actually used. Critical system files (hibernation file, swap file or area, memory dump file) should also be checked systematically.
- Critical configuration files managing the creation of those files should also be protected by a suitable security policy. Anti-virus software should warn against any unsuitable configuration for those files with respect to data persistence.
- Critical software (for example encryption software) should be implemented securely. Before terminating, the physical memory that has been used should be erased securely in order to prevent data remanence. Critical data (such as cryptographic keys) should be locked into memory in order to prevent information becoming available via the swap file or error file (e.g. CORE file under *Linux*). Most high-level programming languages contain suitable primitives that can be used to achieve this.

REFERENCES

[1] Gutmann, P. (1996). Secure deletion of data from magnetic and solid-state memory. Proceedings of the 6th USENIX Security Symposium, pp.77–90.

[2] Gutmann, P. (2001). Data remanence in semiconductor devices. Proceedings of the 10th USENIX Security Symposium, pp.39–54.

[3] Halderman, J.A.; Schoen, S.D.; Heninger, N.; Clarkson, W.; Paul, W.; Calandrino, J.A.; Feldman, A.J.; Appelbaum, J.; Felten, E.W. (2008). Lest we remember: cold boot attacks on encryption keys. Available at http://citp.princeton.edu/memory.

[4] Filiol, E.; Tuccelli, C.; Vuong, T. (2008). Analyse de la mémoire RAM. Récupération de données par le phénomène de rémanence (RAM analysis: data forensics through data persistence). Technical Report ESAT 2008_M2.

[5] http://sourceforge.net/projects/cryptonit/.

[6] Filiol, E. (2004). Strong cryptography armoured computer viruses forbidding code analysis: the BRADLEY virus. Proceedings of the 2005 EICAR Conference. Available at http://vx.netlux.org/lib/aef02.html.

[7] Filiol, E. (2006). Techniques virales avancées. Springer, Collection IRIS, ISBN 978-2-287-33887-8. (An English translation entitled Advanced malware techniques is pending end 2008.)

[8] Riordan, J.; Schneier, B. Environmental key generation towards clueless agents. Lecture Notes In Computer Science, Vol. 1419. http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=746194.

[9] Filiol, E (2007). Formalisation and implementation aspects of K-ary (malicious) codes. EICAR 2007 Special Issue, Broucek V.; Turner, P. eds. Journal in Computer Virology, 3 (2), pp.75–86.

[10] Beaucamps, P., Filiol, E. (2006). On the possibility of practically obfuscating programs. Towards a unified perspective of code protection. Journal in Computer Virology, 2 (4), WTCV'06 Special Issue. Bonfante G.; Marion J.-Y. eds.

FEATURE 2

REVERSING PYTHON MODULES

Aleksander Czarnowski
AVET INS, Poland

The object-oriented programming language Python can be used for many kinds of software development – potentially including malware development. Aleksander Czarnowski believes in being prepared and here he provides a brief overview of how to reverse engineer a Python module.

One might ask: why is there any need to reverse engineer Python scripts? After all, aren't scripts just text files being parsed by an interpreter? In fact, if the parsing process succeeds, Python creates .pyc files from source files. These are in the form of bytecode, which is far from original source.

NOT ONLY BYTECODE

The example presented above is one of four possible situations in which it might be necessary to reverse engineer Python scripts. The other three are: the use of .pyd files; embedding the Python interpreter into a native application written in C/C++; and the use of freeze alike capabilities. I will focus my discussion on .pyc files, but the following paragraphs provide a brief description of each of the other cases:

Essentially, .pyd files are the same as *Windows* DLLs (with a different extension). These files can be imported into a module just like other Python modules (every script is treated as a module in Python). If a file is named 'foo.pyd' it must contain the 'initfoo()' function. The command 'import foo' will then cause Python to search for foo.pyd and attempt to call initfoo() to initialize it.

The Python interpreter may be embedded into a native application for a number of different reasons including as a method of code obfuscation. It would be very easy (in theory at least) to embed Python into a C/C++ application. The simplest method is as follows (for more information see [1]):

```
#include <Python.h>
void runpy(void) {
   Py_Initialize();
   PyRun_SimpleString("print 'hello world from
embedded Python.'");
   Py_Finalize();
}
```

There are several tools that allow a programmer to turn Python scripts into single EXE files. Two popular tools in use today are cx_freeze [2] and py2exe [3]. Internally, these are normal EXE files with an import table – however, keep in mind that this will not tell you much about Python imports or Python code.

I have spent many years using the powerful reverse-engineering tool *IDA Pro*, extending its capabilities with the help of plugins, IDC scripts and Python. I was shocked, therefore, when I attempted to open a .pyc file for analysis, and found that *IDA* did not support the target. With my most powerful tool out of the picture, I had to resort to alternative reverse-engineering methods.

THE PYC FILE STRUCTURE

It turns out that the PYC file structure is quite simple:

	Size (bytes)	Meaning
Magic number	4	The first two bytes of this number tell us which version of Python has been used to compile the file. The second two are 0D0Ah, which are a carriage return and a line feed so that if the file is processed as text it will change and the magic number will be corrupted. (This prevents the file from executing after a copy corruption.)
Modification timestamp	4	This is the Unix modification timestamp of the source file that generated the .pyc so that it can be recompiled if the source changes.
Code object	> 1	This is a marshalled code object which is a Python internal type and is represented as bytecode [4].

More details, such as all the possible magic number values, are included in [5], while [6] and [7] should help explain all the internals.

The .pyc file header can be created by the Module.getPycHeader method:

```
def getPycHeader(self):
 # compile.c uses marshal to write a long directly,
 # with calling the interface that would also
 # generate a 1-byte code to indicate the type of the
 # value. simplest way to get the same effect is
 # to call marshal and then skip the code.
    mtime = os.path.getmtime(self.filename)
    mtime = struct.pack('<i', mtime)
    return self.MAGIC + mtime
```

The MAGIC variable is defined as:
MAGIC = imp.get_magic(). So to determine your Python interpreter magic number you need to enter the following commands:

```
>>> import imp
>>> imp.get_magic()
'\xb3\xf2\r\n'
```

GETTING TO THE MODULE

The beauty of Python is that you can import any module you like as long as it compiles properly. This is not an issue for .pyc files unless the file has been corrupted on disk.

Let's assume our target is called 'sample.pyc'. The following is a sample session from Python interactive mode:

```
ActivePython 2.5.0.0 (ActiveState Software Inc.)
based on Python 2.5 (r25:51908, Mar  9 2007,
17:40:28) [MSC v.1310 32 bit (Intel)] on win 32

Type "help", "copyright", "credits" or "license" for
more information.

>>> dir() #inspect our namespace

['__builtins__', '__doc__', '__name__']

>>> import dis #import Python disassembler -
batteries are really included

>>> import sample #import our pyc.file

>>> dir() #inspect our namespace once again

['__builtins__', '__doc__', '__name__', 'dis',
'sample']

>>> dir(sample) #inspect our target namespace

['__builtins__', '__doc__', '__file__', '__name__',
'foo', 'string']
```

After inspecting the sample.pyc namespace we see there is only one function called 'foo'. To confirm that this is a function we can use the following code:

```
>>> getattr(sample, 'foo')
<function foo at 0x00AE1E70>
```

Now we can use the dis.dis() method to obtain the bytecode of the foo function inside the sample.pyc module (Figure 1).

There is another object in the namespace of our target – 'string'. Let's inspect it, using getattr:

```
>>> getattr(sample,'string')
<module 'string' from 'c:\Program Files\Python25\lib\
string.pyc'>
```

We can see that this is another module that has been imported by our target. Looking at its path we can see it is a standard string module from the Python distribution – but how has this module been imported? We have never run any of the sample.pyc code and a quick inspection of the sample.foo() bytecode reveals no imports. First let's have a look at how the Python code 'import string' is translated into bytecode:

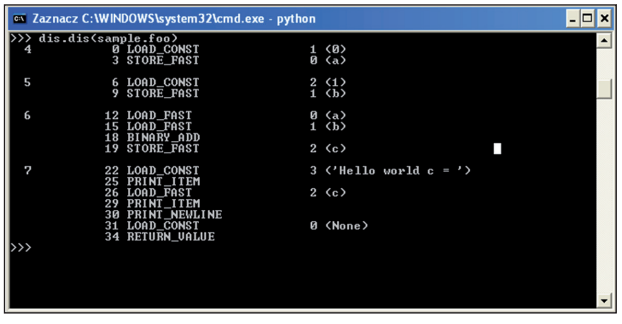


Figure 1: Getting the bytecode of the foo function.

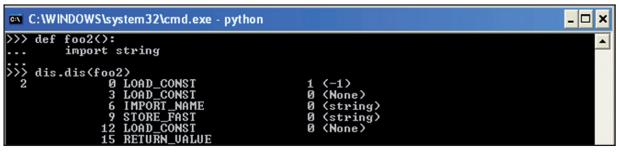


Figure 2: There is no definitive import.

Figure 2 shows that there is no definitive import in our disassembly of sample.foo(). How could this happen? The answer is simple – importing modules means the execution of Python instructions that are not enclosed in classes or functions. So in the case of malware using the import function, this might not be the right solution for disassembling the bytecode. However, we can use the interpreter itself to perform the disassembly. This time we will read the .pyc file by hand and use the marshal module. The marshal module allows bytecode to be loaded from file. As it expects the input to be bytecode, we need to skip the first eight bytes of the .pyc file (the magic number and modtime stamp), as shown in Figure 3.

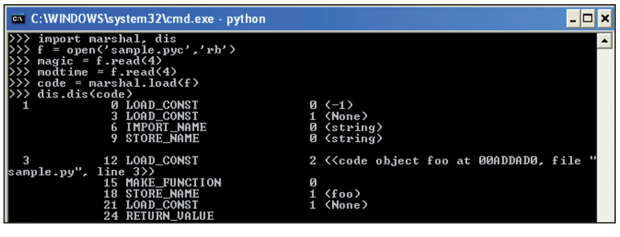


Figure 3: Using the marshal module.

Now we can see our 'import string' instruction in bytecode as well as the creation of the foo() function.

__IMPORT__() AND IMP

Python also allows the importing process to be hooked. Internally, the import instruction calls the __import__() function, which is responsible for all the internal magic that happens during module imports. Also, the imp module can be used for finding and loading modules (imp.find_module and imp.load_module, respectively). This could prove to be helpful during dynamic analysis.

DYNAMIC ANALYSIS

Python comes with a built-in debugger: pdb. Pdb is a module so it is quite simple to use:

```
>>> import pdb
>>> import module_name
>>> pdb.run('module_name.function_name()')
```

Internally, pdb uses sys.settrace to achieve its magic. Like most debuggers, pdb is better suited to cases in which we have access to source code. In fact, when the source code is missing it is quicker to run the script in a controlled environment and trace the system function calls at OS level than to work with pdb. On Win32 systems a set of trusty *SysInternals* tools comes in handy. For larger tasks writing a dedicated sys.settrace handler function would be a possible solution.

REWRITING BYTECODE

Rewriting bytecode is also possible. Byteplay [8] is an interesting project which allows the user to manipulate Python code. The module works with Python versions 2.4 and 2.5. There are also a number of other utilities with similar functionality. Rewriting bytecode could prove useful, for example, in the case of patching .pyc files on the fly.

SUMMARY

The aim of presenting the methods described here was not to provide a definitive reverse-engineering solution but to provide the reader with enough information to find their own path. Python often allows even complex problems to be solved with its built-in functionality. Many of the operations presented here could have been achieved in a simpler manner or using other tools.

I have seen very little information published about Python bytecode. As Python is commonly installed on many Unix/*Linux* systems and is also embedded into several games engines, the ability to understand its bytecode is important as there can be little doubt that it will be targeted by attackers in the future.

REFERENCES

[1] Embedding Python in another application. http://www.python.org/doc/ext/embedding.html.

[2] http://python.net/crew/atuining/cx_Freeze/.

[3] http://www.py2exe.org/.

[4] Internal types: code objects http://docs.python.org/ref/types.html#l2h-143.

[5] http://svn.python.org/view/python/trunk/Python/import.c?view=markup.

[6] http://docs.python.org/lib/compiler.html.

[7] http://svn.python.org/view/python/trunk/Lib/compiler/pycodegen.py?rev=61585&view=markup.

[8] http://code.google.com/p/byteplay/.

FEATURE 3

ADVERTISING DATABASE POISONING

Lysa Myers
McAfee, USA

Adware programs have variously been dressed up as providing anti-phishing protection, intrusion detection capabilities as well as the 'benefit' of targeted advertising, but their presence is still a considerable nuisance to many. Here, Lysa Myers looks into the dubious world of Internet advertising and looks at the effects of programs such as AntiPhorm on adware in general.

Advertising has become an integral part of everyday life – it is almost completely unavoidable. Movies and TV shows have even depicted a fantasy future world in which advertisements appear in our dreams, and on 'smart billboards' which track our every move. Recent developments in targeted advertising have brought the latter scenario increasingly close to a present-day possibility [1], but there are many who find this a completely nightmarish prospect.

In order to obtain this sort of targeted information, advertisers are looking to dig ever deeper into our lives. This naturally raises privacy concerns for those who would prefer not to allow such personal information to get into the hands of complete strangers.

Advertising on the Internet has been a technological testing ground for new information-gathering techniques, and for pushing the boundaries of what is considered acceptable information-gathering behaviour. Almost every sort of network traffic has been used to send advertising content, and now more and more traffic is being monitored in order to tailor such content.

What if there was an effective way to dissuade advertisers from using such invasive techniques? The rise of anti-spyware programs that detect invasive adware as 'potentially unwanted programs' (PUPs) has arguably had some effect on the declining prevalence of advertising software placed on users' computers (see Figure 1). But what can be done when the invasion is being generated from somewhere other than the users' machines?

THE GENESIS OF INTERNET ADVERTISING

The Internet originally started as a place for people to share information and services freely. In order to fund the time and resources needed to maintain a popular website, people needed to come up with ways to make money from the services/information they were providing. Some made their sites subscription-based, charging a fee for their services.

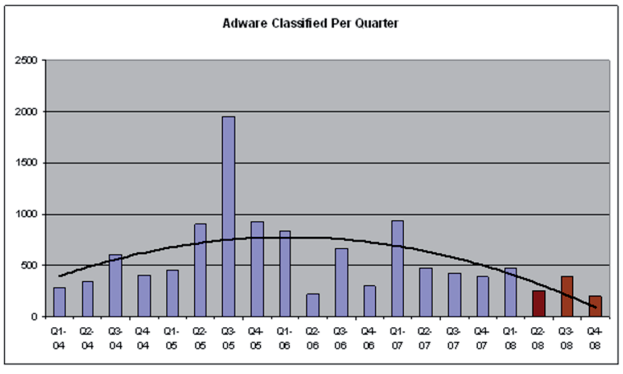


Figure 1: Adware classified per quarter.

Most of the rest turned to advertising revenue as a source of income.

As more sites have turned to using advertising to fund themselves, many are looking to maximize profitability for the advertisers and increase the relevance of advertisements to the user. Demographic information, web-search and email content, as well as Internet surfing habits have all been used to customize advertising content. Demographic information is generally gathered by compulsory registration, whereas surfing habits and email/web-search content is typically gathered without any user interaction.

Many users are offended by what they perceive to be an invasion of their privacy, being obliged to provide personal (even if not personally identifiable) information to an advertiser. Some of these people tried to decrease the incentive for advertisers to gather information this way by providing them with fake information.

BUGMENOT

BugMeNot was a weekend project that was started in August 2003 by an Australian named Guy King [2]. At that point one of the more popular ways for websites to obtain targeted demographic information for advertisers was to require visitors to complete a free registration before allowing them to access content on their sites. This way they could get information about a user's zip code, salary range, gender, date of birth or specific interests, and then sell that information to advertisers.

King created a massive database of information with which to complete the registrations for sites that used this technique, asking the users of *BugMeNot* to help populate this database. A number of rules were put in place to keep people from using it for fraud, or other malicious activities.

Part of the success of *BugMeNot* could be attributed to the developer's decision to make the database accessible via a plugin for *Firefox* – the timing was such that

Firefox was just becoming popular, and many technology and productivity blogs cited the *BugMeNot* plugin as a compelling reason to switch from *Internet Explorer*. Coupling the open-source spirit of the *BugMeNot* database with the *Firefox* browser was a natural match.

In the years since *BugMeNot* became popular, many websites have abandoned compulsory registration. By poisoning the websites' user databases with bogus details on such a wide scale, the information that sites gathered was rendered useless for the purposes of selling to advertisers. The information could not be considered sufficiently trustworthy to ensure advertisers were targeting ads to the right demographic.

BugMeNot was a simple solution for a simple problem, and its story is relatively straightforward. The next generation of data-mining for advertising would be far more intrusive and complex, and its story full of twists and turns.

GOOGLE AND CONTENT MONITORING

In 2000, some years before compulsory registration reached its highest point, *Google* began selling ads based on search keywords. As these were text-based and visually unobtrusive, the ads were generally considered less offensive than the banner ads which were most common at the time. The privacy concerns were few, because search terms were not considered personally identifiable information and because the data that was captured was not intended to leave *Google*.

For those who objected to these keyword ads, two *Firefox* plugins were created, *TrackMeNot* and *SquiggleSR*. These were both designed to create fake searches, to lose the genuine keyword content amongst a flood of automatically generated searches. The traffic from these applications has never been sufficient to motivate any behaviour changes on the part of the search engines.

In April 2004 *Google* introduced *Gmail*, a free web-mail service which boasted 100 times the storage capacity of its leading competitors at the time. To support this service, *Google* included advertising alongside each email viewed, in a form similar to that of the text-based ads that were used in the company's search service. The ads were generated by parsing the content of the email, to ensure relevant content.

This was considered by many to be a serious violation of privacy, as email is ostensibly a private conversation between the sender and the intended recipient(s). Since there were many competitors in the web-mail market, people who found this practice unacceptable generally simply chose an alternative provider.

In the end, *Gmail* was considered a resounding success, and the advertising was viewed by the majority of its users as an acceptable cost for this free service. This success seemed to embolden other advertisers, who saw that users would accept their content being filtered to allow more relevant ad content. However, there were two lessons that these advertisers didn't seem to learn from the success of *Gmail* or the failure of compulsory registration. One was that in order for this to be acceptable, the user had to be given something of significant monetary value. The other was that allowing your advertising database to be accessed by outside parties was considered a greater privacy violation.

DEEP PACKET INSPECTION

In July 2007 *British Telecom* (*BT*) began a test with a company called *Phorm* who used deep packet inspection at the ISP level to gather information on the web-surfing habits of its subscribers and subsequently deliver tailored advertising content. *Phorm* has claimed that it scrubs the content it stores of any personally identifiable information, and that it can also act as an anti-phishing measure as it keeps a list of known phishing sites to prevent users from accessing them.

However, the test was performed in secret, without the knowledge or consent of *BT*'s user-base. It wasn't even widely known that the testing had occurred until the beginning of 2008. This did not set the experiment off on a good note. If this was something that would benefit the user, would the company not have advertised this fact?

Phorm had previously been known by another name (*121media*). In its previous incarnation the company had been associated with an adware application called *Apropos*, which used some of the most devious and sneaky tactics of any such program. The company closed its doors in 2006.

At about the same time as *Phorm* came on the scene, a number of other similar entities began to partner with other ISPs to perform similar data-mining activities. The most well known of these are *NebuAd*, *Front Porch*, *Adzilla* and *Project Rialto*.

The most similar to *Phorm* is *NebuAd*, which has partnered with a number of US-based ISPs, most notably *Charter Communications*. *Adzilla*, like *NebuAd* and *Phorm*, also confines itself strictly to the collection of 'anonymous' web-surfing traffic. It also sells its database to outside parties, in order to serve targeted ad content. Unlike *NebuAd* and *Phorm*, there is little mention anywhere of which ISPs *Adzilla* is partnered with.

Front Porch promotes itself in a significantly different tone. Whereas *Phorm*, *NebuAd* and *Adzilla* all stress the importance of increasing the relevance of ads, *Front Porch* flaunts the ability it gives ISPs to modify their users' Internet experience. It gives the following list of popular uses:

- Redirect subscribers to your portal or a partner's site, regardless of their browser home page settings.
- Offer limited web access to specified subscribers, enabling full access once your conditions are met.
- Redirect subscribers to a partner search engine when they conduct online searches.
- Create a 'walled-garden' of allowed sites for specific subscribers.

Project Rialto has also taken a rather different approach. It is now known as *Kindsight*, and its stated purpose is to provide intrusion detection with its traffic monitoring. The company states that this service is 'funded through an advertising mechanism', providing the users with 'ads on sites that are of interest to the subscriber base'.

THE FIGHT AGAINST DATA MINING

Since the deep packet inspection of companies like *Phorm* was coming from the ISP, and in many areas there are few, if any, competitors for broadband access, a software solution was sought. *AntiPhorm* is a stand-alone program which generates fake web-surfing traffic, intended to bury a user's genuine web-surfing behaviour in a flood of automatically generated traffic. While it was created specifically to work against *Phorm*, it also works with other surfing-trackers and adware applications.

Web surfing is a rather risky business today, with malware infecting legitimate sites as well as more seedy ones. The *AntiPhorm* developers were conscious of this and have taken a variety of steps to minimize any risk to the user caused by additional surfing.

In hidden and text-only modes *AntiPhorm* pre-filters the content it receives to exclude JavaScript, images, video and Flash. It doesn't execute HTML code directly in the browser when in console or hidden mode. Lists of keywords and URLs are both completely customizable, so a user can further restrict what traffic is allowed.

CONCLUSION

The purpose of *AntiPhorm* is to create extraneous and erroneous entries in the advertisers' database, reasonably safely. It seems well suited to this purpose. But will it be as effective as *BugMeNot* in curbing the greater adware trend? While *AntiPhorm* doesn't currently have the benefit of riding the rising popularity of an Internet browser, there are a few outside factors which could work in its favour.

The first is the growing awareness that even information which does not appear to be personally identifiable can be, when taken in context. When a text file containing search keywords from *AOL* was accidentally released on the Internet, it quickly became apparent that information from searching could easily be used to identify the searcher [3]. By ego-surfing, entering addresses or social security numbers, a user's search could easily be mapped to their 'anonymous' numeric ID.

There is also a growing sentiment that the *BT*/*Phorm* tests were illegal, and that the only legally acceptable option is for *Phorm* to be used as an opt-in service rather than opt-out as it is currently set up by most ISPs [4]. This sentiment has been detrimental to *Phorm* in signing up new partners – both *MySpace* and *The Guardian* declined to partner with the company in light of the negative public sentiment [5].

While adware applications have been on the decline recently, their presence is still a considerable nuisance to many. As *AntiPhorm* is a free utility, it may gain popularity with a wider audience who seek to thwart adware thrust upon them by certain freeware vendors [6].

On the other hand, there is one thing that may severely hinder the effectiveness of *AntiPhorm*. Where compulsory registration was used on some of the most popular websites on the Internet, deep packet inspection is used by only a small handful of ISPs at the time of writing. As negative publicity increases for this sort of monitoring, more ISPs are making it opt-in rather than opt-out. It's unlikely to continue to increase in popularity, and it may not ever rise to be the level of nuisance posed by adware.

In effect, the biggest threat to the usefulness of *AntiPhorm*'s advertising database poisoning may simply be that *Phorm* may never gain sufficient popularity. *Phorm* may collapse under the weight of its own bad PR. Perhaps *AntiPhorm* would be best advised to re-brand to appeal to a wider audience of Internet users who are tired of all content monitoring, regardless of the commercial entity behind it.

REFERENCES

[1] http://www.cinematical.com/2007/01/31/minority-reports-intelligent-ads-are-now-science-fact/.

[2] http://www.theage.com.au/news/web/revealed-the-brains-behind-bugmenot/2007/10/08/1191695798003.html?s_cid=rss_technology.

[3] http://en.wikipedia.org/wiki/AOL_search_data_scandal.

[4] http://www.theregister.co.uk/2008/03/17/phorm_fipr_illegal/.

[5] http://www.theregister.co.uk/2008/03/26/guardian_phorm_uturn/.

[6] http://www.geek.com/antiphorm-lite-offers-intelligent-surfing-anonymity/.

PRODUCT REVIEW

SUNBELT SOFTWARE VIPRE ANTIVIRUS + ANTISPYWARE

John Hawes

This month's review product is something quite exciting – a genuinely new anti-malware product emerging from the anti-spyware boom.

Many years ago, anti-virus developers opted to ignore malicious trojan programs, considering them to be outside their remit, and only later did they come to conclude that protection from such threats was a vital part of security. With spyware ignored by many established products, specialist anti-spyware products sprang up to fill the gap. When the wheel turned once again and spyware came to be understood as just another facet of the malware field, most of the leading players in the anti-virus market added anti-spyware functionality to their products. Similarly, players in the anti-spyware market adapted by either buying in or licensing anti-virus technology to complement their own.

Sunbelt Software, meanwhile, whose *CounterSpy* product remains one of the undoubted leaders in the anti-spyware arena, took the more arduous path of developing its own scanning engine to cover the wider range of malicious code. The long-awaited *VIPRE* (*Virus Intrusion Prevention and Recognition Engine*) is the fruit of the company's labours. Currently still in beta, with full release delayed somewhat longer than expected, the product has built up considerable expectations and I was excited to be able to take an early look at its capabilities.

WEB PRESENCE, INFORMATION AND SUPPORT

Sunbelt's online presence, and much of its brand recognition within the security industry, owes a lot to the company's renowned blog – which has become a regular recommendation in 'top 100 blogs' lists including those maintained by *PC World* and *CNET News.com*. The blog (at http://sunbeltblog.blogspot.com/) is run almost single-handedly by the firm's energetic CEO Alex Eckelberry, who keeps up a startlingly regular stream of updates on the latest developments in security, covering new scams and malware techniques, industry and market events, security-related news items (the blog was a pivotal campaign ground in the infamous Julie Amero case), with the occasional off-topic digression into humour and skateboarding pics.

The firm's official website, www.sunbelt-software.com, is a slightly more sober place, but still bright and cheerful and adorned with the hot orange of the company's logo. The

site's front pages are dedicated mostly to promoting the company's product range, which as well as various versions of *CounterSpy* includes a highly respected personal firewall (known as *Kerio* prior to its acquisition by *Sunbelt*), anti-spam and general email security products for home and enterprise users, and a selection of backup, compliance and vulnerability management tools.

Further into the site there is a research subsite offering a range of information and resources, including details of the latest threats, outbreak alerts and a threat database. The database is dominated by spyware but also includes a range of viruses, worms and other types of threat. Product updates and white papers are also provided here, along with access to another product, the *CWSandbox* malware analysis tool. This can be used as an online resource, quickly processing submitted files and providing detailed reports of their behaviours, and is also available as a standalone product for simple and effective analysis of suspect files.

A support section provides a variety of methods for getting assistance, with the usual online form and generic email address complemented by an all-too-rare telephone number. A knowledgebase, which seems fairly well populated, is also provided to solve common issues, and the site hosts an impressive range of busy forums, discussing not only the company's own products but also a selection of other topics of interest to systems and security admins. These services are backed up by a series of news mini-sites providing top stories and comment on various versions of *Windows* as well as *CounterSpy*.

Documentation for the products was a little tricky to find, as I had assumed that the manuals etc. would be included in the support section. However, I eventually turned up a batch of user- and quick-start guides in the products section, which I found were well designed and written in informal, chatty language to minimise the fear factor. Instructions are given based on tasks rather than controls, allowing for easy mastering of important configuration and management jobs. A full user manual did not seem to be available, but this was more than made up for by the excellent inline help I discovered after installing, with the appropriate entry linked

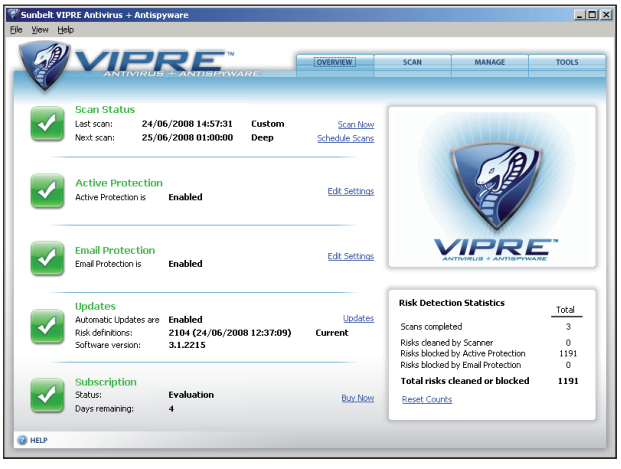
to from just about every area of the product, providing clear and simple guidance on the operation and use of the various functions.

INSTALLATION AND CONFIGURATION

Having prodded around enough, I finally sat down to try out the product itself. The installer package came in at a pretty reasonable 23.5 MB; this compactness, I later discovered, was helped by the product being provided with no detection data at all to begin with, relying instead on an initial update to get everything up to speed. The installation process began along pretty standard lines, with the usual warnings to ensure no other anti-virus software was running, a lengthy EULA, and the selection of install destination before the file-copying process got under way. This seemed reasonably speedy on most systems, taking less than a minute on even a few rather decrepit and underpowered ones. A reboot was required to finalise things.

After the reboot, the completion of a series of setup tasks was required, starting with providing details of any proxy that might be in use before defining the update settings. An initial update could be run manually from here, and options were available for allowing the product to initiate a web connection if required, and to pull down updates at will, with a default timing of every two hours. Next came the 'Active protection' module, the real-time scanner (for which a level of paranoia could be selected), and email scanning, which could be set to monitor particular ports for SMTP traffic if required.

This was followed by the 'ThreatNet' settings, a herd immunity scheme to which users can contribute suspect files if desired, and then the scheduling of scans. This seemed to lack a little granularity and could either be off or running nightly at 1 a.m. – finer tuning of this setting (to allow night hawks to carry on gaming uninterrupted into the small hours) turned out to be available in the interface proper. Then there were some options to integrate with the *Windows Security Center*, and to disable *Windows Defender* if running, then activation and registration options, and finally everything was good to go. Keen users can celebrate the end of this rather lengthy process by



viewing an online demonstration video, in which a smooth voice guides the viewer through the basics of the product's layout.

I decided to explore the product for myself however, and got my first look at the interface itself. It presented a pretty attractive face to the world, adorned with a snazzy snake-on-a-shield logo. The front page is clean and clear, with a list of the major components marked with the standard green tick/red cross to indicate their status, and some simple statistics in one corner. Each section has a link to the appropriate controls page, and a row of tabs along the top provides access to further functions. Even the most inexperienced software user would have no trouble finding their way around.

Most of the settings options lead to the appropriate tab of a unified configuration window, where many of the settings defined during the initial setup process can be adjusted, along with some more in-depth controls for some areas. Granularity in the controls for both on-demand and on-access scanning is fairly reasonable; checking of certain locations, file types and threat types can be switched on or off and response to threats can be either automated or interactive. The on-access scanner can be set to monitor custom file types by extension and even includes a limited intrusion prevention system, which can be set to monitor a range of system areas for signs of unauthorised interference. These monitors 'allow all' by default but can be set to prompt for permission before allowing changes to things like the hosts file, pivotal registry settings and *Internet Explorer* settings.

The tabs along the top give access to a range of extra tools, some management tasks including scheduling jobs, perusing the quarantine and scan and detection histories (detailed logging is available), and the lists of 'always allowed' and 'always blocked' items. The on-demand scanner has its own tab, offering a quick mode and a 'deep' mode, as well as

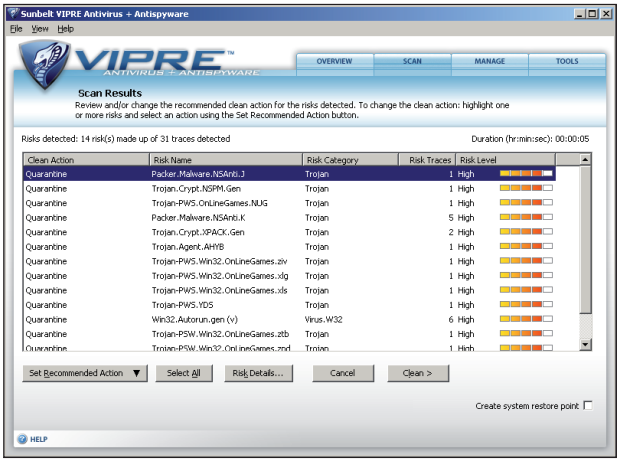
a custom option, and of course can also be operated via a right-click menu entry.

The full set of updates were downloaded fairly speedily but expanded to an impressive 55 MB once installed. After transferring these to my test systems I put the various options to use running some scans over the *VB* sample collections.

SYSTEM PROTECTION AND MALWARE DETECTION

Some initial scans of our clean test sets provided an idea of the scanning speed of the product, which was pretty impressive across the board – perhaps not quite up with the very fastest products measured over the same test sets in recent VB100 comparatives, but some way ahead of most. With the product getting its first glimpse of these large and diverse test sets – which have a habit of tripping up even the most respected products on a regular basis – I expected to see quite a large number of false alarms, but was surprised to find only a tiny number of fairly obscure items flagged as suspicious.

Moving on to the malware sets, scanning across the full range of items produced fewer surprises. Detection rates over the more recent sets of widespread worms and bots were excellent, as was coverage of the collections of trojans and spyware that are currently being compiled from recent reports. File-infecting malware was always going to be more difficult, and detection of some of the older samples was understandably limited, but some of the macro sets were handled impressively. Detection of polymorphic items, including some of the W32/Virut strains riding high in our prevalence reports in recent months, was somewhat patchy, but this is something that *Sunbelt* is working to improve as the product nears its final release, collaborating with certification agencies to ensure more complete coverage.



Although the product was not quite ready for entry in the latest VB100, it looks like a strong contender for achieving certification once it is fully released.

The on-access scanner showed similarly good scanning speeds, reflecting the low scanning overheads experienced during some general playing around on a protected system, and detection rates closely matched those of the on-demand side. A heavy bombardment (attempting to access tens of thousands of infected samples) did seem to overwhelm the product somewhat, bringing up some C++ runtime error messages and leaving the test system pretty crippled, but such an extreme situation is unlikely to be encountered in the real world, and once again the issue should be smoothed out in the final stages of pre-release testing. Turning up the paranoia levels sparked alerts on a wider range of items including the opener tool used for the on-access test, whose behaviour of accessing large numbers of files at once was rightly judged to be a little suspicious.

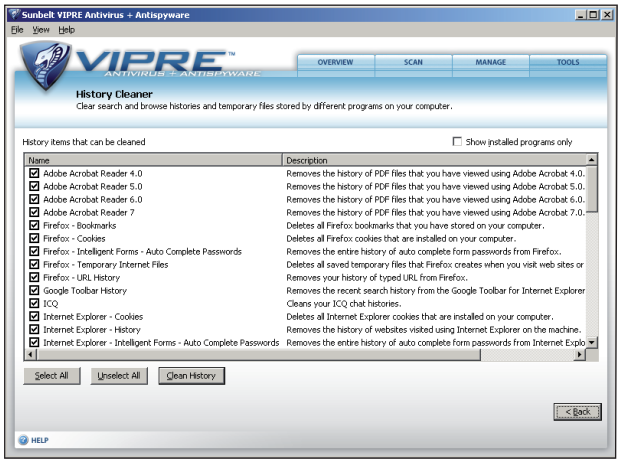
This led me to try out some of the intrusion-prevention monitors available in the advanced options of the 'Active protection' module, which once activated were able to spot and block many of the activities of a selection of new and unknown threats, including changes to the hosts file, installation as startup items, and other common steps in setting up an infection. With all options enabled it pretty much locked the system down, prompting for permission for just about any unexpected execution or action. With the product disabled a handful of other items were installed and, once re-enabled, *VIPRE* showed impressive abilities in the removal and cleanup of some tricky infections.

There was not enough time to carry out fully in-depth testing of the product's various capabilities against a wider range of malware, but I hope to see it appearing in VB100 tests soon. I also hope to be able to review the product again at around the time the suite version emerges, when I will be able to give it a more thorough exercising.

OTHER FUNCTIONALITY

For the moment at least, *VIPRE* provides a bare bones anti-malware system rather than a full suite; integration with *Sunbelt*'s personal firewall is expected soon, along with a corporate version of the product, and it seems likely that some of the company's anti-spam technology will eventually be added too. 'Bare bones' is perhaps a little misleading, as there is in fact considerably more on offer than simple malware detection, blocking and removal.

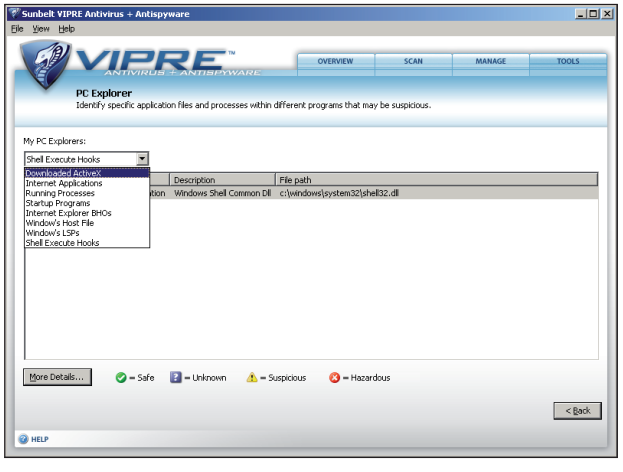
The first entry on the 'Tools' tab is a secure file eraser, which as far the interface is concerned only provides the chance to add an extra deletion option to the *Windows*



Explorer context menu. This promises to shred files securely, beyond the reach of even specialist recovery tools, to ensure confidential data cannot fall into the hands of even the most determined thief. The exact method of deletion is not disclosed, but for most purposes a few levels of random overwriting are a pretty sure bet.

The history cleaner is a rather more complex tool, offering to remove temporary cache files, cookies, browsing history etc. from a pretty exhaustive range of browsers, media players, chat programs and much more besides. These can be configured to show only installed items, and also to leave some products alone, but new products cannot be added manually (presumably updates provided by the vendor can add coverage for extra items and the latest versions of those already included).

The third and last of the extra tools is 'PC Explorer', an even more sophisticated gizmo providing access to a range of low-level information, much of which is often concealed from users in the normal course of things. Lists of running processes, processes launched at startup, installed ActiveX



objects and Browser Helper Objects and the contents of the hosts file, along with several other categories, can be perused, marked as safe if recognised, and more detail on most is available at the click of a button. In stark contrast to the idiot-proof simplicity of the main parts of the interface, this is seriously technical stuff that is likely to be beyond the understanding of the average user, but both fascinating and useful for the more computer-literate. The lack of simple buttons to fix unwanted items would mean that any problems discovered using this tool would require some technical knowledge in order to be corrected manually.

CONCLUSIONS

Having had high expectations of this long-awaited product, *VIPRE* did not disappoint. The design and layout is splendidly clear and useable, the range of features easily accessed and controlled. The protection capabilities are impressive, and will doubtless be even more so once a final release is available. In the area of virus detection, which is fairly new to the company, detection was rather impressive (if not yet up to the same level as the spyware handling), and this looks set to improve in leaps and bounds as the company dedicates more of its time and expertise to the problem.

There are several innovative items in this product, including the limited but potent intrusion prevention options and the string of useful and well-thought-out extra tools. The innovation carries on beyond the technical side of the product to include the availability of a 'home site licence', allowing home users with multiple computers – which is not uncommon these days – to protect all their systems for a single price.

If the next stage of the product's development – rolling in the company's full personal firewall technology to create a full-blown catch-all suite product – can maintain the high standards of design, solidity and usability seen here, it will surely be a force to be reckoned with.

Technical details

Sunbelt VIPRE was variously tested on:

AMD K7, 500 MHz, 512 MB RAM, running *Microsoft Windows XP Professional SP2*.

Intel Pentium 4 1.6 GHz, 512 MB RAM, running *Microsoft Windows XP Professional*.

AMD Athlon64 3800+ dual core, 1 GB RAM, running *Microsoft Windows XP Professional SP2* and *Windows Vista SP1* (32-bit).

AMD Duron 1 GHz laptop, 256 MB RAM, running *Microsoft Windows XP Professional SP2*.

END NOTES & NEWS

The SecureAmsterdam conference on emerging threats takes place 15 July 2008 in Amsterdam, the Netherlands. For details see https://www.isc2.org/cgi-bin/events/information.cgi?event=66.

SANSFIRE 2008 takes place 22–31 July 2008 in Washington, DC, USA. The course schedule for SANSFIRE 2008 features a full line-up in the disciplines of audit, security, management and legal as well as new courses with a focus on penetration testing, malware analysis and removal, and secure coding. For more information see http://www.sans.org/sansfire08/.

The 17th USENIX Security Symposium will take place 28 July to 1 August 2008 in San Jose, CA, USA. A two-day training programme will be followed by a 2.5-day technical programme, which will include refereed papers, invited talks, posters, work-in-progress reports, panel discussions, and birds-of-a-feather sessions. For details see http://www.usenix.org/events/sec08/cfp/.

Black Hat USA 2008 takes place 2–7 August 2008 in Las Vegas, NV, USA. Featuring 40 hands-on training courses and 80 Briefings presentations. This year's Briefings tracks include many updated topics alongside the old favourites including zero-day attacks/defences, bots, application security, deep knowledge and turbo talks. Online registration closes on 31 July. For details see http://www.blackhat.com/.

VB2008 will take place 1–3 October 2008 in Ottawa, Canada. Presentations will cover subjects including: sample sharing, anti-malware testing, automated analysis, rootkits, spam and botnet tracking techniques, corporate policy, business risk and more. Register online at http://www.virusbtn.com/conference/vb2008.

SecTor 2008 takes place 7–8 October 2008 in Toronto, Canada. The conference is an annual IT security education event created by the founders of North American IT security usergroup TASK. For more information see http://sector.ca/.

The 3rd International Conference on Malicious and Unwanted Software (Malware '08) will be held 7–8 October 2008 in Alexandria, VA, USA. The main focus for the conference will be 'the scalability problem'. For more details see http://isiom.wssrl.org/.

Black Hat Japan 2008 takes place 7–10 October 2008 in Tokyo, Japan. Training will take place 7–8 October, with the Black Hat Briefings taking place 9–10 October. For full details see http://www.blackhat.com/.

Net Focus UK 2008 takes place 8–9 October 2008 in Brighton, UK. The event deals with issues of security, personnel, compliance, data privacy, business risk, e-commerce risk and more. For details see https://www.baptie.com/events/show.asp?e=160&xyzzy=2.

The third APWG eCrime Researchers Summit will be held 15–16 October 2008 in Atlanta, GA, USA. eCrime '08 will bring together academic researchers, security practitioners and law enforcement representatives to discuss all aspects of electronic crime and ways to combat it. See http://www.antiphishing.org/ecrimeresearch/.

The SecureLondon Workshop on Computer Forensics will be held 21 October 2008 in London, UK. For further information see https://www.isc2.org/cgi-bin/events/information.cgi?event=58.

RSA Europe 2008 will take place 27–29 October 2008 in London, UK. This year the conference celebrates the influence of Alan Mathison Turing, British cryptographer, mathematician, logician, biologist and 'the father of modern computer science'. For full details including the conference agenda and online registration see http://www.rsaconference.com/2008/Europe/.

CSI 2008 takes place 15–21 November 2008 in National Harbor, MD, USA. Online registration will be available soon at http://www.csiannual.com/.

AVAR 2008 will be held 10–12 December 2008 in New Delhi, India. The 11th Association of anti-Virus Asia Researchers International Conference will be hosted by Quick Heal Technologies Pvt. A call for papers has been issued, with a submission deadline of 15 July. See http://www.aavar.org/avar2008/index.htm.

SUBSCRIPTION RATES

Subscription price for 1 year (12 issues):

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

Corporate rates include a licence for intranet publication.

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

Editorial enquiries, subscription enquiries, orders and payments:
Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England
Tel: +44 (0)1235 555139 Fax: +44 (0)1235 531889
Email: editorial@virusbtn.com Web: http://www.virusbtn.com/

vb Spam supplement

CONTENTS

NEWS & EVENTS

NEW BEST PRACTICES FOR ISPs

Two new sets of best practices for ISPs have been issued by the Messaging Anti-Abuse Working Group (MAAWG), aiming to help block spam sent from botnets and improve the deliverability of consumers' emails.

The first paper, 'MAAWG methods for sharing dynamic IP address space information with others', addresses the issue of blocking botnet spam. MAAWG already recommends that ISPs block traffic from machines on dynamic IP addresses that send email on port 25 (which is likely to be botnet spam) but, since this is not a viable solution for all ISPs, the new paper provides alternative recommendations. The paper describes various ways in which ISPs can share their dynamic space information among one another, thus allowing them the opportunity to reject mail traffic from dynamic address space.

The second paper, 'MAAWG recommendations: email forwarding best practices', proposes methods to help distinguish legitimate customers using a mail forwarding facility from spammers. Many email users have their mail forwarded from one address to another. However, as these addresses receive and forward spam as well as legitimate mail, it is possible for the user's ISP to treat the forwarding service as a spam source and block all incoming mail from it. The MAAWG paper sets out a number of best practices that can be adopted by volume forwarders and the receivers of forwarded mail that will help ensure legitimate mail is delivered. Both papers are available from http://www.maawg.org/.

EVENTS

The 14th general meeting of the Messaging Anti-Abuse Working Group (MAAWG) will be held in Harbour Beach, FL, USA, 22–24 September 2008. See http://www.maawg.org/.

CEAS 2008 will take place 21–22 August 2008 in Mountain View, CA, USA. See http://www.ceas.cc/2008/.

COMMENT

SPEAR PHISHING – ON THE RISE?

Paul Baccas
Sophos, UK

The spam traps at *SophosLabs* receive millions of emails every day. We have complex internal systems that process these emails. The majority of the emails are classified as spam automatically, and as such they may never be seen by a human (yes, researchers are human). As a researcher therefore, I tend only to see spam that is causing our customers a problem – in other words, emails that are not being classified automatically or that are not being received by our spamtraps.

Recently, we have seen an increase in targeted phishing, or spear phishing campaigns. These campaigns are not being seen by traditional spamtraps, though they are being seen by our customer base.

SPEAR PHISHING

Spear phishing is the targeted phishing of users. By pretending to be an internal employee – often an IT administrator – the phisher gains access to local credentials. Once the bad guys have local credentials they may use that information for a variety of purposes:

- To hack the box in order to install malware (spambots etc.)
- To hack other users' information
- To phish other users in the company
- To gain further information from the customer

The issues of security information reuse mean that once someone has obtained one password then they may have access to several others.

For example:

- A phisher sends an email to joe.doe@company.x under the pretence of being an IT administrator and asks for the user's username and password.
- Joe Doe enters his details into a website. Username: jdoe & Password: Lakers

- This information tells the phisher the format of Company X usernames (first initial followed by surname) and that the company does not enforce strong passwords (and therefore they are susceptible to a dictionary search).
- The information also leads the phisher to suspect that Joe is a basketball fan – often secondary security information is sports related.

EXAMPLE

In Figure 1 we see a typical text-based phish requesting the recipient's email username, password etc. In the 'To' field is the address of a member of staff or student at Oxford Brookes University, and there were a large number of addresses in the CC field (including my work email address – and Brookes is not even my alma mater). The 'From' addresses is forged. However, it only takes one person to give away their details for a phishing campaign to be successful.

Once the phisher has one piece of personally identified information (PII) it makes it easier for them to gain other pieces.

Figure 1: Example of a spear phish from PureMessage quarantine sent to me.

ON THE RISE?

Phishers phish for economic reasons. Both direct phishing of bank details and spear phishing for personal information ultimately generate an income for the phisher. However, direct phishing is becoming less profitable for a number of reasons, which can broadly be categorised into social and technological:

- Social reasons:
  - User education. Education has raised the level of awareness among users of the dangers of phishing, and as a result users are becoming more wary of the emails they receive and less likely to be tricked.
  - Bank effort. Many banks alert their customers when a phishing attack is known to be targeting their organization. Some are also beginning to change their style of communication with their customers to avoid confusion with phishing emails – for example by not including any links to their sites and instead requiring the customer to enter the bank's URL manually or to bookmark the site.
- Technological reasons:
  - Browser enhancements and add-ons that flag suspected or known phishing sites.
  - Proactive anti-phishing rules incorporated into anti-spam products.

In my opinion it is the last of these that has had the greatest impact on the profitability of the more traditional phishing methods. As a result, phishers are moving away from direct phishing and concentrating their efforts instead on spear phishing or on another more lucrative business.

Spear phishing is less efficient than direct phishing for a number of reasons:

- A smaller volume of phishes are sent.
- Better spam filtering means that the number of phishes that reach the recipients may be very low.
- More effort is required to extract the profit.
- User education means that users are wary of giving away personal information such as that requested in spear phishes (although they are more likely to expect emails from and reveal information to IT staff).

But for the phisher, the plus side of spear phishing is that the lower volume of emails and their targeted nature mean the phish have lower visibility to spam filtering software, and as a result spear phishing is becoming more popular among phishers.

CONCLUSION

Spam is nearly all about the perceived financial reward for the spammers. Phishing is all about the economic reward, and as long as one person falls victim to the scam, phishers will keep on phishing. As one modus operandi becomes unprofitable another will open up. You can guarantee that somewhere in the world a phisher is thinking, à la Cuba Gooding Jr., 'Show me the money'.