## 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

---

### CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>COMMENT</td>
<td>Threat intelligence sharing: tying one hand behind our backs</td>
</tr>
<tr>
<td>3</td>
<td>ANNOUNCEMENT</td>
<td>The shape of things to come</td>
</tr>
<tr>
<td>4</td>
<td>MALWARE ANALYSES</td>
<td>The curse of Necurs, part 1</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>More fast or more dirty?</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Tofsee botnet</td>
</tr>
<tr>
<td>14</td>
<td>TECHNICAL FEATURE</td>
<td>Back to VBA</td>
</tr>
<tr>
<td>22</td>
<td>COMMENTARY</td>
<td>Is the IT security industry up to the new challenges to come?</td>
</tr>
<tr>
<td>24</td>
<td>SPOTLIGHT</td>
<td>Greetz from academe: no place to Hyde</td>
</tr>
<tr>
<td>25</td>
<td>END NOTES &amp; NEWS</td>
<td></td>
</tr>
</tbody>
</table>
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 ENISA¹, 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) protocol² and FS-ISAC³ 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 arms⁴ 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.

² http://www.mitre.org/capabilities/cybersecurity/partnership.
³ https://www.fsisac.com/.
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 apply1 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 one. It 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’, the ‘Group’ registry value to ‘Boot Bus Extender’, 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 Extender’ 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.
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 ‘!/784Uy554NYSY84IOK/’ in the file. As always, the packer will decrypt and load the payload in memory. Figure 1 shows the decryption routine.

```
.text:00483c76  loc_44bb7c76:          ; CODE XREF: sub_44bb2a0+471f]
.text:00483c77  80 80 00 FF FF+ mov ecx, [ebp+var_100]  ; ecx = payload size - 1
.text:00483c82  80 85 70 FF FF+ mov eax, [ebp+var_90]  ; eax = index
.text:00483c83  36 C1 cmp eax, ecx  ; check whether the decryption has finished or not
.text:00483c84  0f 8f 0a 0c 0c jg loc_44be44
.text:00483c8a  ; irrelevant code hidden
.text:00483c9a  80 85 70 FF FF+ mov eax, [ebp+var_90]  ; eax = index
.text:00483ca7  27 ; irrelevant code hidden
.text:00483c9f  ;
.text:00483cda  loc_44be40:          ; CODE XREF: sub_44bb2a0+317f]
.text:00483cd2  80 85 70 FF FF+ mov [ebp+var_100], eax  ; var_100 = index
.text:00483cd8  ; irrelevant code hidden
.text:00483d0a  80 85 70 FF FF+ mov eax, [ebp+var_90]  ; eax = index
.text:00483d16  60 cdq
.text:00483d21  7f 70 44 idiv [ebp+var_5c]  ; edx = key index - index % 0x45
.text:00483d32  80 85 70 FF FF+ mov eax, [ebp+var_90]  ; eax = index
.text:00483d42  ; irrelevant code hidden
.text:00483d48  ;
.text:00483d58  loc_44be68:          ; CODE XREF: sub_44bb2a0+35df]
.text:00483d5e  80 85 70 FF FF+ mov [ebp+var_100], edx  ; var_100 = key index
.text:00483d66  ; irrelevant code hidden
.text:00483d76  80 85 70 FF FF+ mov eax, [ebp+var_90]  ; eax = index
.text:00483d86  ; irrelevant code hidden
.text:00483d8c  ;
.text:00483d92  loc_44bf05:          ; CODE XREF: sub_44bb2a0+39tf]
.text:00483d98  80 55 0c mov edx, [ebp+var_44]  ; ecx = address of encrypted payload buffer
.text:00483da2  88 48 0c mov ecx, [edx+90h]  ; eax = address of the key buffer
.text:00483da8  80 42 0c mov eax, [edx+8ch]  ; eax = address of the key buffer
.text:00483dab  80 95 7a FE FF+ mov edx, [ebp+var_10c]  ; edx = key index
.text:00483db2  88 0a 18 mov al, [eax+edx]  ; get 1 byte from key buffer
.text:00483db8  88 95 7a FE FF+ mov edx, [ebp+var_10b]  ; edx = index
.text:00483dbd  32 0a 11 xor al, [ecx+edx]  ; decrypt 1 byte
.text:00483dc3  80 0a 11 mov [ecx+edx], al  ; save decrypted data
.text:00483dcf  ; irrelevant code hidden
.text:00483dd8  80 80 70 FF FF+ mov ecx, [ebp+var_90]  ; ecx = address of encrypted payload buffer
.text:00483dea  88 01 00 00 00 mov eax, 1  ; ecx = key index
.text:00483e07  88 80 0a 00 00 add ecx, eax
.text:00483e13  80 80 70 FF FF+ mov [ebp+var_90], ecx  ; increase index by 1
.text:00483e1d  jmp loc_44bf7a
```

Figure 1: The decryption routine.
Figure 2: Part of the request parameter list.

```
lea eax, [esp+18Ch+String1]
push [esp+18Ch+String1] ; nCount
push eax ; lpClassName
push esi ; hWnd
	text:00401479  FF 00 00 00
	text:0040147A  FF 15 64 71 40+ call ds:GetClassNameA
	text:00401480  8D 4C 24 0C lea ecx, [esp+1Ch+String1]
text:00401484  68 4C 70 55 0C push offset aleframe ; "IEFrame"
text:00401489  57 push ecx ; lpString1
text:0040148A  FF 15 34 70 40+ call ds:1strcmpIn
	text:00401490  86 C0 test eax, eax
text:00401492  75 34 jnz short loc_4014C8
	text:00401494  8D 54 24 08 lea edx, [esp+1Ch+dwProcessId]
text:00401498  52 push edx ; lpdwProcessId
text:00401499  56 push esi ; hWnd
text:0040149A  FF 15 50 71 40+ call ds:GetWindowThreadProcessId
text:0040149D  88 00 88 A7 55+ mov ecx, dwProcessId
text:004014A0  88 44 24 08 mov eax, [esp+18Ch+dwProcessId]
text:004014A4  8B 01 jmp eax, ecx
	text:004014A7  75 10 jnz short loc_4014C8
	text:004014AE  56 push esi ; hWnd
text:004014AF  89 35 BC A7 55+ mov hWndParent, esi
text:004014B0  FF 15 60 71 40+ call ds:SetForegroundWindow
```

Figure 3: Check whether the target window belongs to the process created by itself.
Figure 4: Simulates the Tab key (several times) and the Enter key.

Figure 5: ‘feed.php’ forwards the request 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:

<table>
<thead>
<tr>
<th>MD5</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>39412490E7221EA8A2C5125CC8CFC447</td>
<td>morefastsearch.com</td>
</tr>
<tr>
<td>F6CEA38DF990A0DCF73167D4E359728B</td>
<td>bzmp3.com</td>
</tr>
<tr>
<td>D86DEEEFD8AF29390F408E684BD64E5F1</td>
<td>bzmp3.com</td>
</tr>
<tr>
<td>15ED911F307A8E005F6A6BDDD58A0C3</td>
<td>firstsearchnow.com</td>
</tr>
<tr>
<td>C1A1F9DC8449B34F8BEF0F6EB937C8F</td>
<td>webfindpage.com</td>
</tr>
<tr>
<td>F4A2705067AD1405D3354D1CAA0EC855</td>
<td>zbeemp3.com</td>
</tr>
</tbody>
</table>

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.
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.

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:

```
```

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.

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).

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

<table>
<thead>
<tr>
<th>%Type%</th>
<th>%Name</th>
<th>UpdateCount</th>
<th>LastUpdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>listen_srv</td>
<td>7 2013-12-18 00:43:16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-work_srv</td>
<td>6 2013-11-25 12:42:33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*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.

<table>
<thead>
<tr>
<th>%Type%</th>
<th>%Name</th>
<th>UpdateCount</th>
<th>LastUpdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-12</td>
<td>5 2014-01-10 06:43:53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-18</td>
<td>3 2013-12-19 12:43:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-19</td>
<td>2 2013-12-11 12:42:38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-14</td>
<td>3 2013-12-10 06:42:14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-4</td>
<td>3 2013-12-04 12:42:24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-6</td>
<td>5 2013-11-30 06:42:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-5</td>
<td>5 2013-11-30 06:42:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-11</td>
<td>5 2013-07-22 12:06:42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-12</td>
<td>5 2013-07-22 12:06:42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-3</td>
<td>1 2013-07-22 12:06:42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-2</td>
<td>1 2013-07-22 12:06:42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-1</td>
<td>1 2013-07-22 12:06:42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-7</td>
<td>1 2013-07-22 12:06:42</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*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.

*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.

*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.
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.

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

```vb
#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
```

![Figure 1: Overview of infection method.](image)

**INFECTION PROCESS**

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

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.

![Figure 4: Luring the user into enabling macro execution.](image)

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:

```assembly
push 082899612h ; 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
```

```assembly
```
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.
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.

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:

- 3716e5ba7ed966228e79036698419a78a9583b62: crypsola.exe
- c72d5c35e8aaa366b457e622ab235641c06376a: IE_Explorer.exe
- 14de27f59db24129073f6b35641f61a179d013dfe: 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(\(i\))) 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:

```autoit
EXECUTE ( BINARYTOSTRING (  "0x2449664745575456426a73202620426a68722841736328537472696e67466e420282466764684753668a67736957722c2024692c20312929202b203929" ) )
```

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

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

Finally, the EXECUTE(\(i\)) 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.

![Figure 7: Encrypted shellcode and its decoder.](image)

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.

![Figure 8: RC4 decoder shellcode implementation.](image)

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.

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\’.
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: Reaseen 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

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].

![Figure 9: Process injector shellcode invoked from the AutoIt script.](image1)

![Figure 10: String building.](image2)

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:

```vb
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,

---

1 37f6e5ba7cd966228e79036698419a78a9583b62
this embedded content is saved to the file deepweb.txt in the temporary directory. The file contains an ASCII representation of the payload EXE.

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

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:

$SKFAPMBTBj7463539079213644 = "SxdMcxwLe18682537269213644"
$ARJXYJBAUV842669899213644 = "hhojVVeBo196456697179213644"
LOCAL $MLFJUEIDLE = EXECUTE ( BINARYTOSTRING (AVNVLTILJTHBER ( ".v224a4a3152505341522150434/52430600405752433`000.040.0.0.20474c4/50572a434c06023/52562331274d3/3042070.04003b0007" ) ) )
$PAUVSHBGNI9389858899213644 = "wrAHosOjXb20608857089213644"
$EKFSLEBMHU10353018890213644 = "MckeIpOQqn97180529213644"

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

FILEINSTALL ( "kFxIFMBTBjgn9675177345409009.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).

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).

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

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

- dopline.ru
- terra-araucanica.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


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**


**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. 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**


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 Infosecureurope 2014 exhibition and conference will be held 29 April to 1 May 2014 in London, UK. For details see http://www.infosec.org/.

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.


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/.

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/.


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:

Tel: +44 (0)1235 555139  Fax: +44 (0)1865 543153
Email: editorial@virusbtn.com  Web: http://www.virusbtn.com/

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

This publication has been registered with the Copyright Clearance Centre Ltd. Consent is given for copying of articles for personal or internal use, or for personal use of specific clients. The consent is given on the condition that the copier pays through the Centre the per-copy fee stated below.

Tel: +44 (0)1235 555139. /2014/$0.00+2.50. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form without the prior written permission of the publishers.

ADVISORY BOARD

Pavel Baudis, AVAST Software, Czech Republic
Dr John Graham-Cumming, CloudFlare, UK
Shimon Gruper, NovaSpark, Israel
Dmitry Gryaznov, McAfee, USA
Joe Hartmann, Microsoft, USA
Dr Jan Hruska, Sophos, UK
Jeannette Jarvis, McAfe, USA
Jakub Kaminski, Microsoft, Australia
Jimmy Kuo, Independent researcher, USA
Chris Lewis, Spamhaus Technology, Canada
Costin Raiu, Kaspersky Lab, Romania
Roel Schouwenberg, Kaspersky Lab, USA
Roger Thompson, ICSA Labs, USA
Joseph Wells, Independent researcher, USA