C16406984
Arthur Coll

Q1. How was the RAM captured?

The first thing I did was identify the image type using:

```
vol.py -f "Win7 RAM Capture.raw" imageinfo
```

This identified the image as Windows 7 Service Pack 1, following this I appended all commands with --profile Win7SP1x86_23418

After this I used the pslist command to identify all the currently non hidden processes.

```
vol.py  -f "Win7 RAM Capture.raw" --profile Win7SP1x86_23418 pslist
```

Within this there was an entry for dumpit.exe which is a popular memory dumping application, this is then corroborated later in the MFT file tables and in the internet search history. Therefore it’s reasonably safe to assume dumpit.exe was used to capture the ram.

Q2. State how many processes were running at the time. (i.e. in the OS process list)

To measure this I used pslist to output all the current processes and then used powershell Measure-Object -line to output the current number of lines which was 47 which minus the 2 header lines gives us 45 processes. This was then verified with psxview to verify there was no hidden processes.

```
(python vol.py psx -f "Win7 RAM Capture.raw" --profile=Win7SP1x86_23418 pslist |
Measure-Object -line).Lines
```

```
(python vol.py psx -f "Win7 RAM Capture.raw" --profile=Win7SP1x86_23418 psxview |
Measure-Object -line).Lines
```

Q3. How many files were open at the time? How many jpeg files were open at the time?

To count the number of files opened I used the filescan command which outputs the currently opened files in memory and their details. I then counted it similar to above.

```
 (python vol.py  -f "Win7 RAM Capture.raw" --profile=Win7SP1x86_23418 filescan  |
Measure-Object -line).Lines
```

The total number of files open was 6809 at the time of the dump.

(python vol.py -f "Win7 RAM Capture.raw" --profile=Win7SP1x86_23418 filescan | findstr ".jpg" | Measure-Object -line).Lines

After this I ran the same command but used the findstr function to filter for only files with the extension of “.jpg” which produced 25, notably there was also a file open with the “.jpeg” extension making the total number of JPG files open arguably 26.

Q4. What Searches appear in the internet history? What browser was used in the search?

A number of searches appear in the history of both chrome and internet explorer. Chrome history was browsed via the chromehistory plugin which parses the SQLite database.

python vol.py --plugins=plugins/ -f "Win7 RAM Capture.raw" chromehistory

The main items within were searches related to Irish Tourism such as Madison hotel and Carrick a rede rope bridge, dumpit memory tool, hotmail and yahoo mail. The two email accounts were donnie.duckie@hotmail.com and donnie.duckie@rocketmail.com
Notably there was visits to a website that allows you to draw on pictures suggesting some sort of basic stenography potentially.

python vol.py -f "Win7 RAM Capture.raw" --profile=Win7SP1x86_23418 iehistory

Internet Explorer was much less used but there was evidence from the iehistory plugin that it was used to download chrome, view some files, visit MSN and other microsoft services primarily. Under the account donald@ie.msn.com/

Q5. A file called duck.gif was downloaded. When did this happen? What program do you think was used to open the file? Give reasons for your answer.

Piping the chrome history into the findstr command outputs the url of a wordpress site called say no to crack where it seems duck.gif was downloaded from.

However it’s unlikely it was viewed in chrome because that is not the default handler for gif’s, when use the same piping of the findstr on internet explorers history we can clearly see he has opened duck.gif a couple of times from the local filesystem as indicated by the file:// protocol in the URL. This can then be verified later when we dump the memory for IE Explorer which shows it was accessed.

Q6. Choose a likely process and dump its memory to a file. Examine the dump with strings. Search for ASCII strings and UTF16 strings. Find something and point it out.

First I extracted and dumped the memory of the IE explorer process 3692 using memdump as follows.

```
python vol.py  -f "Win7 RAM Capture.raw" --profile=Win7SP1x86_23418 memdump -p 3692 -D dumpfiles
```

I then analyzed it with the strings tools from sysinternals, I used the -a & -u flags respectively to output UTF & ASCII. There was a lot of information contained within, search history data I already looked at, information on the process itself such as the fact it was working in compatibility mode and information that confirmed the previous findings. I couldn't find anything we hadn't already covered.

```
strings64.exe -a 3692.dmp > outputAsci.txt
strings64.exe -u 3692.dmp > outputUni.txt
```