Patrycjusz Zdzichowski, Michal Sadlon, Teemu Uolevi Väisänen
Alvaro Botas Munoz, Karina Filipczak

Anti-Forensic Study
Executive Summary

The use of anti-forensic techniques in and on IT systems is common practice for advanced and persistent actors, particularly in the contexts of targeted attacks or efforts by organised criminals to erase digital traces. This might include tampering with log files, using wiping or ‘cleaning’ tools, deploying rootkits, using hidden data storage areas, or even deploying traps to be activated in the course of a later investigation. Therefore it is necessary to describe the state of the art in anti-forensic techniques in order to identify and elaborate on potential detection or mitigation techniques for practitioners in the field.

This paper is primarily intended to be read by forensic specialists and information security professionals. It describes in detail how modern anti-forensic tools work and how to mitigate them. We focus mainly on Windows operating systems, but some of the tools described can be used on Linux as well.

The first and the second sections of this report detail the proposed classification, with the description of each technique. The third section describes several anti-forensic techniques and tries to find possible mitigation scenarios. It focuses on techniques such as timestamp manipulation, memory pollution or data hiding, which allow attackers to remain undetected and hide their malicious activity for as long as possible. Some methods can break disk or memory acquisition tools, and this unequivocally indicates a use of anti-forensic techniques.

In our research, we have shown how powerful anti-forensic tools can be in the hands of skilled criminals. There are numerous resources that teach readers how to exploit or break the forensic process, but we lack guidance in how to protect against them. Scenario analysis for this study demonstrates that live acquisition and memory analysis are areas of common ground in which to remain one step ahead. It is particularly beneficial to use strings, especially against memory images or volatility plugins like cmdscan or consoles.

Some malicious activities do not touch the hard drive and will only become visible in the memory. However, memory analysis is not fully reliable without double-checking the output and the final result that is the memory image, as several tools might tamper with the evidence before analysis. Therefore, there is a strong need for custom imaging tools and specific procedures to avoid the situation where we are left with evidence that is of no use to the case we are working on. There are some hints in this paper on how to prepare for such moments.

This study is based on a Request for Support to the NATO Cooperative Cyber Defence Centre of Excellence (NATO CCD COE) dated 25 March 2014. This request was submitted through the NATO CCD COE Steering Committee and was approved for implementation.
About the NATO CCD COE

The NATO Cooperative Cyber Defence Centre of Excellence (NATO CCD COE) is an international military organisation accredited in 2008 by NATO’s North Atlantic Council as a ‘Centre of Excellence’. Located in Tallinn, Estonia, the Centre is currently supported by the Czech Republic, Estonia, France, Germany, Hungary, Italy, Latvia, Lithuania, the Netherlands, Poland, Slovakia, Spain, the United Kingdom and the USA as Sponsoring Nations, and by Austria as a Contributing Participant. The Centre is neither part of NATO’s command or force structure, nor is it funded by NATO. However, it is part of a wider framework supporting NATO Command Arrangements.

NATO CCD COE’s mission is to enhance capability, cooperation and information-sharing between NATO, NATO member states, and NATO’s partner countries in the area of cyber defence by virtue of research, education and consultation. The Centre has taken a NATO-oriented interdisciplinary approach to its key activities, including academic research on selected topics relevant to the cyber domain from the legal, policy, strategic, doctrinal and/or technical perspectives, providing education and training, organising conferences, workshops and cyber defence exercises, and offering consultations upon request. For more information visit http://www.ccdcoe.org.
Table of Contents

Executive Summary ................................................................................................................................. 3
About the NATO CCD COE .................................................................................................................. 4
Table of Contents ................................................................................................................................... 5
1 Introduction ........................................................................................................................................... 6
2 Classification of anti-forensic techniques ............................................................................................ 8
  2.1 Data ................................................................................................................................................... 8
  2.2 Memory related anti-forensic techniques ......................................................................................... 9
  2.3 Digital Forensic Tools ................................................................................................................... 10
3 Anti-forensic techniques ...................................................................................................................... 11
  3.1 Timestamps manipulation .............................................................................................................. 11
  3.2 DoS attacks against forensics investigation tools ........................................................................... 20
  3.3 Data hiding ....................................................................................................................................... 25
  3.4 Usbkill ............................................................................................................................................ 30
  3.5 Live Linux Distributions ................................................................................................................. 31
  3.6 Memory anti-forensics .................................................................................................................. 37
  3.7 Microsoft anti-forensic settings ..................................................................................................... 59
4 Conclusions .......................................................................................................................................... 62
References ............................................................................................................................................... 63
1 Introduction

Traditionally, digital forensic science has been defined as:

‘the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorised actions shown to be disruptive to planned operations’. [1]

Later, this definition evolved to a more proactive approach to not only collecting and analysing the data related to an incident, but also to prevent further escalation. The new approach emerged as Proactive Digital Forensic Component, defined as ‘the ability to proactively collect, trigger an event, and preserve and analyse evidence to identify an incident as it occurs’. [2] As a result, the collection phase comes before preservation in order to trigger preliminary analysis. At this point, if the incident is identified, analysts have more time to react, and perpetrators have limited time to hide their tracks. The outcome of those very first steps would be a preliminary report based on gathered information from proactively obtained evidence.

Digital forensic investigations can be divided into several sub-branches, based on the technical aspects. The classical branch is called Computer Forensics and contains: storage media (usually done post-mortem); physical memory forensic (very often used in preliminary analysis); and electronic devices such as Personal Digital Assistants (PDA), radars, detectors and surveillance electronics. The second sub-branch represents Mobile Phone and Smartphone Forensic. This component requires a different type of knowledge, equipment and investigation approach. Some information can be read only from removed EEPROM and other types of flash memory chips, and that leads to yet another sub-branch of Digital Forensics – Chip Forensics. Finally, lawful interception and malware analysis can be united under the term Network Forensics. As we can see, Digital Forensics deals with very heterogeneous objects and data.

For several years, this has been a growing field in industry and academia. Digital Forensic Science has also been used in many trials by lawyers to describe the chain of events. As always, the perpetrators have taken advantage of their victims to steal their data. Those activities violate privacy and are considered crimes. Despite the fact that the field is advancing quite quickly, it is almost impossible to keep track of it. Most of the results from ongoing investigations are off the record and not available to the public. The main reason for that is to hide the available techniques and solutions from criminals, but it has also made the lesson-learning process tougher, as sharing results with forensic community is becoming something that not everyone is willing, or even allowed, to do.

In order to reconstruct the sequence of events for any criminal and unauthorised actions, recover deleted files, or gain intelligence about a user’s computer, digital forensic investigators should follow a specifically structured workflow called the forensic process. We can divide it into the following phases:

- Verification – focuses on detection of an incident from indicators and evaluating its type.
- Preparation – involves the arrangement of techniques, tools, required authorisation documents and management support.
- Approach strategy – develops a procedure to maximise the collection of evidence while minimising the impact on the environment.
- Preservation – entails the isolation, preservation and securing of the state of evidence.
- Collection – which involves recording of the physical scene and duplicating evidence using scientifically derived and proven methods.
- Examination – focuses on identifying and locating potential evidence relating to suspected crime.
• Analysis – involves steps like timeline analysis, media analysis, data recovery and string search. The main goal is to determine significance, reconstruct fragments of data, and draw preliminary conclusions based on evidence found.

• Reporting – the most important step, involves summarising and providing an explanation of conclusions. The details of the examination should allow others to repeat the steps.

• Returning evidence – ensuring that any physical or digital evidence is returned to its proper owner. It also includes procedures regarding how and what criminal evidence must be removed.

All those phases can be simplified into four main steps: seizure, acquisition, analysis and reporting. We can say that this approach enhances the science of forensics, because it provides a basis for analysing new digital technology while at the same time giving a joint framework for law enforcement and the judicial system to conceivably work within a court of law.

On the other side, methods have also developed to evade the forensic process, either by the perpetrator hiding the way the attack is carried out, or to protect the privacy of the user. These techniques are called counter-forensic or anti-forensic techniques. We can define them as ‘any attempts to compromise the availability or usefulness of evidence to the forensic process’.[1] Thus anti-forensics is a general term for any techniques intended to delay, complicate, subvert or inhibit forensic techniques for finding evidence. It is also a way of conducting criminal hacking that can be summed up as: making it as hard as possible to find you and impossible to prove that it was you. Usually, in the world of modern operating systems and the way they manage information, data, and events, this ‘evidence’ is generally pretty easy to find, especially when the attacker does not expect any action against him. However, for very sophisticated computer related crimes, multiple techniques might be used to increase the complexity of the analysis of the discovered media, even if the suspect has been identified. Therefore, one of the options for computer villains is to block the deposition or collection of any traces by erasing or destroying them before they can be acquired as evidence. It should be noted that these definitions do not take into account using anti-forensics methods for ensuring the privacy of people’s personal data.

The number of papers on protecting against anti-forensic methods is greatly outnumbered by the number of websites, forums and other resources that teach readers how to exploit the forensic process. Therefore, it would seem that attackers are working harder to subvert the system than the scientific community is working to strengthen forensics.

The paper is structured as follows: The next section details the proposed classification, with the description of each technique. The third section describes several anti-forensic techniques and tries to find possible mitigation scenarios. It focuses on techniques that allow attackers remain undetected as long as possible or hide their activity. Some methods can break disk or memory acquisition tools, but this unequivocally indicates a use of anti-forensic techniques. Finally, the fourth section states final conclusions.
2 Classification of anti-forensic techniques

Based on the fact that the forensic process relies on the analysis of the data collected during the evidence acquisition phase, we can categorise anti-forensics techniques that aim at the whole process or in some cases just parts of it. From this starting point, we can predict whether the attack will corrupt the desired final result of the investigation. In many situations, forensic investigators can still demonstrate that they were able to work with some evidence, even without knowing the specific content of that evidence.

2.1 Data

This category involves techniques that directly affect information stored on volatile media. It includes metadata, the location of the data, the data content used to perform the attack, and/or the stolen information that needs to be extracted.

2.1.1 Storage

All methods centred around storage focus on making the acquisition process as difficult and time-consuming as possible. One of the best examples is the use of custom RAID arrays. Building custom arrays makes offline disk acquisition practically pointless, because later on investigators will not be able to reconstruct them to retrieve relevant data.

2.1.2 Hiding

The primary goal of hiding techniques is to store sensitive data in such a way as it will not likely be found or decrypted. This part includes methods like steganography, or in- or out-band hiding. Steganography is all about hiding data inside data. File formats used may vary depending on the amount of information we want to conceal. In-band hiding focuses on alternative file streams, file-system journal logs or reserved, unallocated sectors. The general rule is that hidden data must not break the format of the specification. On the other hand, we have out-band hiding that can use slack space beyond the end of a partition, sectors marked as bad, slack space at the end of files or the Host Protected Area.

2.1.3 Source elimination

Sometimes prevention is the best cure. This is particularly true when it comes to anti-forensics. Rather than put all sorts of energy into covering up a trace after it gets generated, the most practical route is one that never produces the evidence to begin with. A good example could be the use of live Linux distributions. All activity is done during a live Linux session run from an external device such as a USB or optical disk that leaves no trace on the physical hard drive. Of course, this works only if it is mounted in read-only mode.

2.1.4 Fabrication

Data fabrication is a genuinely devious strategy. Its goal is to flood the forensic analyst with false positives and bogus leads so that a victim ends up spending most of his time chasing his tail. You essentially create a huge mess and let the forensic analyst clean it up. For example, if a forensic analyst is going to try and identify an intruder using file checksums, then you simply alter as many files on the volume as possible. It also involves techniques like the addition of known files, string decoration, and false audit trials.

2.1.5 Physical/logical destruction

In this method, the attacker wants to buy time by leaving as little useful evidence as possible. Data destruction helps to limit the amount of forensic evidence generated by disposing of data securely after it is no longer needed or by sabotaging data structures used by forensic tools. This could be as simple as wiping the memory
buffers used by a program, or it could repeatedly involve overwriting to turn a cluster of data on disk into a random series of bytes. In some cases, data transformation can be used as a form of data destruction.

Physical destruction of potentially evidence-providing storage is merely physically harming the devices such that the evidence on them may be damaged or corrupted.

2.1.6 Saturation

The best example to explain data saturation would be a compression bomb. This is compressed content which extracts to a size much larger than expected; to put it simply, incorrect handling of highly compressed data. This technique can result in various denial-of-service conditions, such as memory, CPU and free disk space exhaustion.

2.1.7 Virtualisation

Although the growth of virtualisation technologies brings benefits to more efficient use of computer resources, it also carries a consequence of using virtual machines as anti-forensic tools. Some products allow the user to install virtualisation software on USB flash drives or other remote disk storages (FTP servers, clouds etc.) and to run applications directly from those devices without ever ‘touching’ the physical storage of the computer. In this case, no traces of the attacker activities that were performed using the virtual machine will be left on the file system of the physical machine.

2.2 Memory related anti-forensic techniques

Random Access Memory (RAM) is the bridge between the CPU, storage devices, and operating system. Nearly everything of interest that has ever happened on a modern computer has traversed the RAM. From files to network connections to registry hives to running malware, a wealth of data is available for analysis. That is why recently we have seen strong emphasis on research anti-forensic tools in this field.

2.2.1 Artefacts hiding

In this group, we can distinguish techniques that prevent the initial acquisition of volatile evidence from the machine or just sensitive parts of it that the attacker wants to hide. The most popular method to break memory acquisition is ‘One-Byte Modification’ where an additional kernel driver changes one byte of the size in _DISPATCHER_HEADER of PsIdleProcess, pool tag in _POOL_HEADER of PsInitialSystemProcess and MajorOperatingSystemVersion in the PE header of Windows kernel. That leads to a situation where an acquired image is missing a tremendous amount of information required for memory analysis. Of course, this can raise suspicions; therefore, it is more advisable to use more sophisticated methods such as the interception of NtWriteFile() calls. An anti-forensic tool called Dementia facilitates this technique to hide processes or communication artefacts in the memory.

2.2.2 Pollution

The other approach is to make the memory image analysis phase as unreliable as possible is by putting additional artefacts into memory. Bogus information like fake processes, file strings or TCP connection attributes can make the analysis phase very time-consuming and adds doubt to the evidence in legal proceedings. One of the examples in here is the Attention-Deficit-Disorder (ADD) tool presented by Jake Williams and Alissa Torres during ShmooCon Conference in 2014 [3].
2.3 Digital Forensic Tools

Digital Forensic investigators rely on one or two tools to follow their investigations. The reliance on a small number of tools is because of user confidence, costs of required training, and the community’s standardised tools. Tool risk types fall into three categories: denial of service attacks, failure to validate data and fragile heuristics. In an anti-forensic domain, the primary objective is to break forensic process and to produce fake data that will have no value in a court of law; therefore, we can distinguish two types of methods to achieve this goal: detection and exploitation.

2.3.1 Detection

Detection covers all the techniques that react to the presence of Digital Forensic Tools (DFT) and either modify output or block the acquisition process. One example is the detection of a host in ‘promiscuous’ mode. Since many forensics systems capture all packets on the LAN rather than just those addressed to the host, these systems are detected when they respond to various kinds of malformed IP packets. Another example is countering forensic analysis with SMART. This is Self-Monitoring, Analysis and Reporting Technology that is built into most hard drives today and has the ability to report the total number of power cycles, log the high temperatures that the drive has reached, log the total amount of time that a hard drive has been in use, and record other manufacturer-determined attributes. User programs read those counters and cannot be reset. Although the SMART specification can implement a DISABLE command, experimentation indicates that the few drives that actually implement the DISABLE command continue to keep track of the time-in-use and power cycle count and make this information available to the next power cycle. Anti-forensic tools can read SMART counters to detect attempts at forensic analysis and alter their behaviour accordingly. For example, an increase in Power_On_Minutes might indicate that the computer's hard drive has been imaged.

2.3.2 Exploitation

Exploitation includes any anti-forensic techniques that successfully facilitate vulnerabilities or weaknesses of Digital Forensic Tools (DFT). This category refers to traditional software flaws such as buffer overflow (Encase CVE-2007-4037) or fragile heuristics, which is based on the fact that DFT frequently needs to determine the type of file or data objects to allow for efficient processing. For example, a forensic examiner may try to save time by omitting the contents of executable files from searches. Many tools determine file type by just consulting the file's extension and the first few bytes of the file's contents (the ‘magic number’). Attackers who know the heuristics that a DFT uses for identifying data can exploit them.
3 Anti-forensic techniques

3.1 Timestamps manipulation

Interacting with most file systems is like walking in snow – every activity will leave footprints. Digital forensics is the art of analysing these artefacts. Timestamps help an analyst create a timeline of events and profile hacker behaviour. If any suspicious file is encountered, an investigator will search for other files with similar time attributes. For various reasons, an attacker wants to make it hard for a forensic analyst to determine the actions that he or she took. One way to do this is to change the time values of the touched files. If the timestamps of the malware have been changed, this can make it difficult to identify a suspicious file as well as to determine when the malware arrived on the system. An anti-forensic technique that manipulates filesystem timestamps is called timestomping.

There are two sets of timestamps that are tracked in the MFT, $STANDARD_INFORMATION and $FILE_NAME. Both of them track four timestamps – Modified, Access, Created and MFT entry modified (MACE or MACB timestamps). The $STANDARD_INFORMATION timestamps are the ones normally viewed in Windows Explorer as well as some forensic tools.

3.1.1 Testing environment.

Operating Systems:
1. Windows 7 Enterprise x64 SP1, 1 CPU, 8GB, physical machine
2. Windows Server 2008 x86 SP2, 2 CPU, 2GB, VMware Workstation 11.1 VM

Used forensic tools:
1. AccessData FTK Imager 3.2.0.0
2. ExtractUsnJrnl 1.0.0.1
3.LogFileParser 2.0.0.20
4. Mft2Csv 2.0.0.26
5. UsnJrnl2Csv 1.0.0.6
6. Sleuthkit-4.1.3
7. Volatility Framework 2.4
8. INDXParse.py

3.1.2 Timestomp

https://www.offensive-security.com/metasploit-unleashed/timestomp/

Timestomp is a filesystem timestamp manipulation tool. It is a program that alters all four NTFS file times. It modifies only the MACE times stored in a file’s $STANDARD_INFORMATION attribute and those not in the $FILE_NAME attribute, thus, leaving some indicator of suspicious activity. Timestomp utility syntax is:

timestomp #filename# [options]

#filename# – the name of the file you wish to modify (you may need to surround the full path in ‘ ’)

Options:
- m #date# M, set the ‘last written’ time of the file
- a #date# A, set the ‘last accessed’ time of the file
- c #date# C, set the ‘created’ time of the file
- e #date# E, set the ‘mft entry modified’ time of the file
- z #date# set all four attributes (MACE) of the file
- v display UTC MACE values of the file
- r set the MACE timestamps recursively on a directory
Time/date format: ‘DayOfWeek MMDDYYYY HH:MM:SS [AM|PM]’

3.1.2.1 Results

Testing scenario:

- Create a text file on the external USB device with NTFS filesystem and change timestamps

Status of the test_timestomp.txt file before changing the timestamps:

```
Then run command: timestomp e:\test_timestomp.txt -z ‘Sunday 10/10/2010 10:10:00 AM’
```
From the output, $STANDARD_INFORMATION time values are changed according to the command, but $FILE_NAME time values are unchanged.

### 3.1.2.2 Analysis and possible mitigation techniques

NTFS $LogFile and change journal ($USnJrnl) files were exported after the timestomp application testing.

Output from NTFS change journal related to timestomp application activity:

<table>
<thead>
<tr>
<th>FileName</th>
<th>USN</th>
<th>Timestamp</th>
<th>Reason</th>
<th>MFTReference</th>
<th>FileAttributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_timestomp.txt</td>
<td>6736</td>
<td>2015-08-17 13:19:54.397:2414</td>
<td>BASIC_INFO_CHANGE</td>
<td>37</td>
<td>archive</td>
</tr>
<tr>
<td>test_timestomp.txt</td>
<td>6832</td>
<td>2015-08-17 13:19:54.397:2414</td>
<td>BASIC_INFO_CHANGE+CLOSE</td>
<td>37</td>
<td>archive</td>
</tr>
</tbody>
</table>

From this log, an analyst does not see what kind of change was exacted on the specific file but there is evidence of when it happened. An identifier BASIC_INFO_CHANGE means that the file attributes and/or the time stamps were changed. The CLOSE identifier indicates that this is the final modification made to the file in this series of operations.

Snippets from the NTFS $LogFile output:

- The first part only confirms previous changes visible also in NTFS change journal.
In the NTFS $LogFile, it is possible to find changed time values. As can be seen below, there is an 
UpdateResidentValue operation with modified $STANDARD_INFORMATION values. The names of the 
timestamp that are used in the output of LogFile Parser tool are different from the commonly used MACE (MACB) terminology. The timestamps refer to:

CTime means File Create Time;
ATime means File Modified Time;
MTime means MFT Entry modified Time;
RTime means File Last Access Time.

<table>
<thead>
<tr>
<th>If_RedoOperation</th>
<th>lf_FileName</th>
<th>lf_SI_CTime</th>
<th>lf_SI_ATime</th>
<th>lf_SI_MTime</th>
<th>lf_SI_RTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateResidentValue</td>
<td>test_timestamp.txt</td>
<td>2010-10-10 10:10:00.000:0000</td>
<td>2010-10-10 10:10:00.000:0000</td>
<td>2010-10-10 10:10:00.000:0000</td>
<td>2010-10-10 10:10:00.000:0000</td>
</tr>
</tbody>
</table>

It is possible to detect the timestamp tool not only from the NTFS $LogFile or change journal, but also from a timestamp itself. One of the ways to tell if file time backdating has occurred on a Windows machine is to examine the filename times compared to the times stored in $STANDARD_INFORMATION. Generally, attackers do this only to programs they are trying to hide in the System32 or similar system directories. Those directories and files would be a great place to start. An analyst should look to see if the $FILE_NAME time occurs after the $STANDARD_INFORMATION Creation Time, as this would indicate an anomaly. Therefore, another possible validation technique is to determine if the $STANDARD_INFORMATION MACE values are older than $FILE_NAME MACE values.

Another tool for helping the analyst figure out that timestamps are changed by the timestomp tool is NTFS timestamp resolution. NTFS timestamps have 100 nanosecond precision. Therefore it is also suspicious if a timestamp has for example value 2010-10-10 10:10:00.000:0000 (as shown above in the entry from $LogFile). It means that after manipulation with timestomp, the updated timestamp loses its resolution beyond seconds.

### 3.1.3 SetMace

https://github.com/jschicht/SetMace

SetMace is an advanced filesystem timestamp manipulation tool, originally inspired by timestomp. Tool is capable of modifying the timestamps in both the $STANDARD_INFORMATION and the $FILE_NAME attributes within the MFT. It is also possible to modify timestamps within volume shadow copy. This program modifies timestamps by direct disk access. For this reason, it needs to acquire exclusive disk access and the utility operates with elevated privileges from the user space, unmounts the target volume, and carries out its modifications. But acquiring direct disk access to the volume from which Windows is booted is not possible from user space in Windows since Windows Vista,[4] and so performing timestamps manipulating operations with SetMace on such a volume needs a different approach; for example by booting a portable Windows OS installation or by attaching the disk to some other Windows machine. So for Windows OS since Vista SetMace cannot access the physical disk of any system volume, but it can access the physical disk of nonsystem volumes. On Windows XP OS SetMace can reset timestamps on a file on any volume as XP’s security model did not restrict access to the physical disk.[5]

There is no step that explicitly changes the $FILE_NAME timestamps and there is no function that allows a user to easily change these values. To avoid this problem, SetMace moves the file to a directory on the same volume. This process causes the $FILE_NAME timestamps to be updated to reflect those from the $STANDARD_INFORMATION attribute.[6]
Steps which SetMace follows to alter timestamps:

1. Change the $STANDARD_INFORMATION timestamps of the file.
2. Create randomly named directory on the same directory.
3. Move the file to the newly-created directory
4. Change the $STANDARD_INFORMATION timestamps of the file
5. Move the file back to the original location
6. Change the $STANDARD_INFORMATION timestamps of the file.
7. Delete the newly-created directory.

SetMace utility syntax is similar to timestomp. There are 5 parameters:

- Parameter 1 is input/target file. It must be a full path.
- Parameter 2 is determining which timestamp to update:
  - ‘-m’ = LastWriteTime
  - ‘-a’ = LastAccessTime
  - ‘-c’ = CreationTime
  - ‘-e’ = ChangeTime (in $MFT)
  - ‘-z’ = all 4
- Parameter 3 is the wanted new timestamp. Format must be strictly followed like YYYY:MM:DD:HH:MM:SS:NS. Timestamps are written as UTC.
- Parameter 4 determines if $STANDARD_INFORMATION or $FILE_NAME attribute or both should be modified.
  - ‘-si’ will only update timestamps in $STANDARD_INFORMATION (4 timestamps), or just LastWriteTime, LastAccessTime and CreationTime (3 timestamps) for non-NTFS;
  - ‘-fn’ will only update timestamps in $FILE_NAME;
  - ‘-x’ will update timestamps in both $FILE_NAME and $STANDARD_INFORMATION
- Parameter 5 is optional
  - ‘-shadow’ will activate shadow copy mode

3.1.3.1 Results

Testing scenario:

- create text file on the external USB device with NTFS filesystem (this enable physical access to volume) and change timestamps (both $FILE_NAME and $STANDARD_INFORMATION attribute time values)
Status of the test_setmace.txt file before changing the timestamps:

```
C:\tools\leuthkit-4.1.3-win32\bin\istat -o 0864 \.\\PHYSICALDRIVE1 38
MFT Entry Header Values:
Entry: 38  Sequence: 2
File Sequence Number: 16817962
Allocated File
Links: 2

$STANDARD_INFORMATION Attribute Values:
Flags: archive
Owner ID: 0
Security ID: 263
Last User Journal Update Sequence Number: 6256
File Modified: 2015-08-17 12:05:56 (FILE Daylight Time)
MFT Modified: 2015-08-17 12:05:56 (FILE Daylight Time)
Accessed: 2015-08-17 12:05:56 (FILE Daylight Time)

$FILE_NAME Attribute Values:
Name: test_setmace.txt

Parent MFT Entry: 5  Sequence: 5
Allocated Size: 0  Actual Size: 0
Created: 2015-08-17 11:54:49 (FILE Daylight Time)
File Modified: 2015-08-17 11:54:49 (FILE Daylight Time)
MFT Modified: 2015-08-17 11:54:49 (FILE Daylight Time)
Accessed: 2015-08-17 11:54:49 (FILE Daylight Time)

$OBJECT_ID Attribute Values:
Object ID: 0000e056-5008-83be-11e5-44a358cf32e7

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-3) Name: N/A Resident size: 98
Type: $FILE_NAME (64-2) Name: N/A Resident size: 98
Type: $OBJECT_ID (64-4) Name: N/A Resident size: 16
Type: $DATA (128-5) Name: N/A Non-Resident size: 3024 init_size: 3024
```

Then run command: `SetMace64.exe e:\test_setmace.txt -z '2010:10:10:10:00:00:768:1234' –x`

```
C:\tools\leuthkit-4.1.3-win32\bin\istat -o 0864 \.\\PHYSICALDRIVE1 38
MFT Entry Header Values:
Entry: 38  Sequence: 2
File Sequence Number: 16817962
Allocated File
Links: 2

$STANDARD_INFORMATION Attribute Values:
Flags: archive
Owner ID: 0
Security ID: 263
Last User Journal Update Sequence Number: 6256
File Modified: 2010-10-10 13:00:00 (FILE Daylight Time)
MFT Modified: 2010-10-10 13:00:00 (FILE Daylight Time)
Accessed: 2010-10-10 13:00:00 (FILE Daylight Time)

$FILE_NAME Attribute Values:
Name: test_setmace.txt

Parent MFT Entry: 5  Sequence: 5
Allocated Size: 0  Actual Size: 0
Created: 2010-10-10 13:00:00 (FILE Daylight Time)
File Modified: 2010-10-10 13:00:00 (FILE Daylight Time)
MFT Modified: 2010-10-10 13:00:00 (FILE Daylight Time)
Accessed: 2010-10-10 13:00:00 (FILE Daylight Time)

$OBJECT_ID Attribute Values:
Object ID: 0000e056-5008-83be-11e5-44a358cf32e7

Attributes:
Type: $STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
Type: $FILE_NAME (48-3) Name: N/A Resident size: 98
Type: $FILE_NAME (64-2) Name: N/A Resident size: 98
Type: $OBJECT_ID (64-4) Name: N/A Resident size: 16
Type: $DATA (128-5) Name: N/A Non-Resident size: 3024 init_size: 3024
```

- be aware that timezone in the testing environment was UTC+3

From the output it is evident that both MFT entry attributes time values are changed.
3.1.4 Analysis and possible mitigation techniques

This newer method of timestamp manipulation is harder to detect. In order to do so an examiner cannot rely on analyzing $UsnJrnl or $LogFile. Since this tool uses direct physical access there are no NTFS $LogFile or USN journal entries that show changes.

In the NTFS $LogFile, an analyst cannot find changed time values like the timestomp application. Instead, there is a different piece of evidence in the NTFS change log file for filesystem from where the SetMace application was launched. SetMace needs access to the sectorio.sys file (or sectorio64.sys for 64-bit OS) that is the kernel mode driver reading and writing to sectors on the underlying physical disk.

In the table below (output from an NTFS change log) you can see the operations on the filesystem where setmace64.exe was used. This is also evidence of using SetMace and the same events are created with every SetMace use.

<table>
<thead>
<tr>
<th>File Name</th>
<th>USN</th>
<th>Timestamp</th>
<th>Reason</th>
<th>MFTReference</th>
<th>File Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sectorio64.sys</td>
<td>2061164992</td>
<td>2015-08-14 13:17:04.625:2289</td>
<td>FILE_CREATE</td>
<td>65716</td>
<td>archive</td>
</tr>
<tr>
<td>sectorio64.sys</td>
<td>2061165080</td>
<td>2015-08-14 13:17:04.625:2289</td>
<td>DATA_EXTEND+FILE_CREATE</td>
<td>65716</td>
<td>archive</td>
</tr>
<tr>
<td>sectorio64.sys</td>
<td>2061165168</td>
<td>2015-08-14 13:17:04.635:2290</td>
<td>CLOSE+DATA_EXTEND+FILE_CREATE</td>
<td>65716</td>
<td>archive</td>
</tr>
<tr>
<td>sectorio64.sys</td>
<td>2061165872</td>
<td>2015-08-14 13:17:06.037:7310</td>
<td>CLOSE+FILE_DELETE</td>
<td>65716</td>
<td>archive</td>
</tr>
<tr>
<td>sectorio64.sys</td>
<td>2061166200</td>
<td>2015-08-14 13:17:07.210:2326</td>
<td>FILE_CREATE</td>
<td>65716</td>
<td>archive</td>
</tr>
<tr>
<td>sectorio64.sys</td>
<td>2061166288</td>
<td>2015-08-14 13:17:07.210:2326</td>
<td>DATA_EXTEND+FILE_CREATE</td>
<td>65716</td>
<td>archive</td>
</tr>
<tr>
<td>sectorio64.sys</td>
<td>2061166376</td>
<td>2015-08-14 13:17:07.230:2327</td>
<td>CLOSE+DATA_EXTEND+FILE_CREATE</td>
<td>65716</td>
<td>archive</td>
</tr>
<tr>
<td>sectorio64.sys</td>
<td>2061166464</td>
<td>2015-08-14 13:17:07.280:2327</td>
<td>CLOSE+FILE_DELETE</td>
<td>65716</td>
<td>archive</td>
</tr>
</tbody>
</table>

3.1.4.1 Evidence in the memory

An analyst acquired the physical memory more than one hour after using the SetMace utility. As expected, it is not possible to detect the SetMace process. An analyst can use the volatility plugin cmdscan to find commands that were typed or executed.

```
**************************************************
Command/process: conhost.exe Pid: 4884
CommandDictionary: 0x38edd0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 3 LastAdded: 2 LastDisplayed: 2
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x64
Cmd #0 @ 0x38dae0: cd \n
Cmd #1 @ 0x38adc0: cd tools\SetMace

Cmd #2 @ 0x3942e0: SetMace64.exe e:\test_setmace.txt -z '2010:10:10:10:10:00:768:1234' -x

Cmd #15 @ 0x320158: 9
Cmd #16 @ 0x320158: 9

**********************************************************
```

The output from volatility cmdscan:
Another way is to use the volatility plugin consoles which do not only print the commands typed, but it collects the entire screen buffer.

The output from the volatility consoles:

```
**************************************************
CommandHistory: 0x38edd0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 3 LastAdded: 2 LastDisplayed: 2
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x64
Cmd #0 at 0x38dae0: cd \
Cmd #1 at 0x38adc0: cd tools\SetMace
Cmd #2 at 0x3942e0: SetMace64.exe e:\test_file.txt -z '2010:10:10:10:10:00:768:1234' -x
----
Screen 0x36b6d0 X:100 Y:400
Dump:
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>cd \\nC:\>cd tools\SetMace
C:\tools\SetMace>SetMace64.exe e:\ test_setmace.txt -z '2010:10:10:10:10:00:768:1234' -x
Starting SetMace by Joakim Schicht
Version 1.0.0.16
Target filename: test_file.txt
Target fileref: 36
Target MFT record offset: 0x00000000C0009000
Parent filename: .
Parent fileref: 5
Parent MFT record offset: 0x00000000C0001400
--
--
--
Attempting write to physical disk without driver
Success dismounting e:
Success writing timestamps
File system cache cleared in RAM
Job took 3.38 seconds
**************************************************
```

It is possible to extract and verify the MFT entries of the files from the memory. To do that, an analyst can run the volatility plugin mftparser and then look for a particular file. Distinctions between an MFT entry from memory and filesystem is that timestamps in memory are changed immediately. We cannot rely on comparing timestamps from MFT entries in memory to those on disk in an effort to detect timestomping ([5], p. 492-493). The timestomping program has its own MFT entry, which the examiner might find in the memory.

Example of the output from the volatility mftparser plugin:
Another evidence of the timestomping program is the Shimcache registry entry that can also be found in memory. The Shimcache registry key is part of the Application Compatibility Database and contains a path for executable and the last modified timestamp from the $STANDARD_INFORMATION attribute of the MFT entry. The Shimcache volatility plugin lists all the entries from the memory.

Example of the output from volatility shimcache plugin:

<table>
<thead>
<tr>
<th>Last Modified</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-08-06 11:56:20 UTC+0000</td>
<td>??\C:\Af\SetMace\SetMace.exe</td>
</tr>
<tr>
<td>2015-07-29 17:57:10 UTC+0000</td>
<td>??\C:\Windows\SoftwareDistribution\Download\Install\mpyschk.exe</td>
</tr>
</tbody>
</table>

3.1.5 Key remarks

Final assumptions resulting from previous analysis:

- $STANDARD_INFORMATION time is before $FILE_NAME time.
- Nanoseconds values are all zeroes (works for timestomp tool).
- File change time should be greater than parent directory birth time (a file cannot be placed in a directory that does not yet exist).[7]

Be aware that some inconsistencies may occur, but not often – the false positive ratio is low enough for further examination of the results manually.

There are some basics methods/artefacts that could help to detect or reveal the use of timestamp manipulation tool:

- Prefetch file – the timestomping application creates a .pf file after it executes, which an analyst can use to show that it ran. Be aware that this functionality is not enabled by default on Windows Server OS and workstation using SSD drives.
- UserAssist – this registry key stores a list of programs executed on a Windows machine, complete with running count and the last execution time. Unfortunately, it does not work for programs launched from the command line.
- Volume Shadow Copies – used to compare the current timestamps of the files with the timestamps from the past, evidence of historical fragments.
To sum up, timestamp manipulation is a popular anti-forensic technique that can make forensic investigation challenging and time consuming. There is no automated way that would easily reveal this kind of AF technique, and not all tools are mentioned in this chapter. There are also some GUI tools that work on the timestamp application principle, but the methods of detection are the same.

### 3.2 DoS attacks against forensics investigation tools

The investigation process and tools can be attacked via modified log, audio, picture, video, office, pdf, or email storage files. This section describes a few anti-forensics techniques that could be used for DoS attacks against forensics tools and presents some ideas for mitigating them.

#### 3.2.1 Introduction

An XML bomb (or billion laughs) [9] is designed to overcome servers and users’ computers by consuming resources of XML parsers.

A ZIP bomb (also known as a zip of death or decompression bomb), is a malicious archive file designed to crash or render useless the program or system reading it.[20] Such bombs are often deployed to disable antivirus (AV) software in order to create an opening for more traditional viruses, but they could be used to attack a forensics tools and make investigations harder.

A zip bomb allows the program to work as intended, but the archive is carefully crafted so that unpacking it (e.g., by AV scanner in order to scan for viruses) requires inordinate amounts of time, disk space and/or memory. It is claimed in [10] that most modern AV programs can detect if a file is a zip bomb and avoid unpacking it, but on 26th August 2015 the detection ratio of 42.zip in Virustotal.com was 22/56, and on 9th September 2015 21/57.[11] There are public guidelines and webpages [12][13][14] for helping to create ZIP bombs, so they can be easily used by script-kiddies.

A fork bomb [15] (or rabbit virus or wabbit) is a process that continually replicates itself to deplete available system resources.

#### 3.2.2 Attack scenarios

Attacks against systems using ZIP, XML and fork bombs could include but are not limited to the following:

- **Adversary could create new ZIP bombs or insert or embed them inside other documents, such as .doc or .pdf. The purpose would be to make an AV scanner to consume too much CPU, disk space and memory, making the system unusable or forcing the user to shut down the AV scanner.**
- **Adversary could create XML bombs to crash the XML parser of the browser. By storing html and other pages in systems and inserting XML bombs into them, it could be possible to make the investigation slower.**
- **Adversary could craft a huge figure that has a small size but also consumes memory when opened.**
- **Adversary could encrypt, decode, obfuscate, and/or compress a short amount of sensitive data and add this then to a ZIP bomb to, for example, hide information for later use.**
- **Adversary could encrypt, decode, obfuscate, and/or compress a short amount of sensitive data, add this to a ZIP bomb, split the ZIP bomb into several packets, and send these packets to one or many dropzones. The purpose would be the exfiltration of data.**
- **Adversary could insert ZIP bombs into emails.**
- Adversary could create several similar type of files that look the same and store them to same location. Some of the files would be ZIP bombs and some would not. The purpose could be to slow the investigation.
- Adversary creates fork bombs that start running only when the live incident response investigation is detected, consuming the resources of the computer and making the incident response harder.

In all scenarios, the adversary could use obfuscation techniques such as modifying headers of the files. It is also possible that the adversary could combine different DoS techniques.

3.2.3 Examples

3.2.3.1 Crafted images

As described in [16] and in [17], it is possible that an adversary crafts a png figure showing certain uniform colours with a width and height of thousands of pixels each. The information is stored in a file by using png format’s compression feature. For example a compressed size of 19K x 19K pixels image in only about 44,024 bytes (44kB), but the file inflates to 1GB when viewed in 24-bit colour mode.

The following commands present information about example file picture-1G-19000x19000.png which was downloaded from [16].

```bash
$ sha256sum picture-1G-19000x19000.png
50d5587d5b1d62382450959ee202862808e7a944603ccf3b8c9006fbl02d1a9 picture-1G-19000x19000.png
$ file picture-1G-19000x19000.png
picture-1G-19000x19000.png: PNG image data, 19000 x 19000, 1-bit colormap, non-interlaced
$ exiftool picture-1G-19000x19000.png
ExifTool Version Number : 9.69
Directory : .
File Size : 43 kB
File Modification Date/Time : 2015:09:09 13:20:36+03:00
File Access Date/Time : 2015:09:09 13:20:42+03:00
File Inode Change Date/Time : 2015:09:09 13:20:38+03:00
File Permissions : rw-rw-r--
File Type : PNG
MIME Type : image/png
Image Width : 19000
Image Height : 19000
Bit Depth : 1
Color Type : Palette
Compression : Deflate/Inflate
Filter : Adaptive
Interlace : Noninterlaced
Palette : 255 0 0
Image Size : 19000x19000
$ less picture-1G-19000x19000.png
picture-1G-19000x19000.png PNG 19000x19000 19000x19000+0+0 8-bit PseudoClass 1c 44KB 0.000u 0:00.000
$ hexdump picture-1G-19000x19000.png
0000000 5089 474e 0a0d 0a1a 0000 0d00 4849 5244...

There seem to be differences between software handling such images. When opening the file with ImageMagick 3.12.2 or Chromium 43.0.2357.130 browser of 64-bit Ubuntu 14.10 running in Lenovo T540p with 16GB of memory, the use of CPU and RAM goes up for short time. See display.im6 in figure below for ImageMagick. The highest values were not recorded.
Opening picture-1G-19000x19000.png with ImageMagick.
The image was not shown properly in Image Viewer 3.12.2 using the Mozilla Firefox 39.0 browser. Opening was not tested with a machine that would not have had enough resources. Notice that user names are hidden in the figure above. It is possible that such files could do harm to investigation machines by, for example, crashing certain image viewers and at least making the investigation harder. Notice that the image size could be basically anything, but we did not do tests with larger images.

3.2.3.2 42.zip

One infamous example is 42.zip [18][sha256sum: db6981082063dbb4bad89d27c41fobe8ed9e4a97b36661c0945b77a6896b0948], which structure and sizes are presented in the Table below. The size of the compressed file is 42 kilobytes, but after fully unpacking everything, the whole volume size is 4.5 petabytes.

Structure of 42.zip.

<table>
<thead>
<tr>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.zip</td>
</tr>
<tr>
<td>+= lib 0.zip ... lib f.zip</td>
</tr>
<tr>
<td>+= book 0.zip ... book f.zip</td>
</tr>
<tr>
<td>+= chapter 0.zip ... chapter f.zip</td>
</tr>
<tr>
<td>+= doc 0.zip ... doc f.zip</td>
</tr>
<tr>
<td>+= page 0.zip ... page f.zip</td>
</tr>
<tr>
<td>+-- 0.dll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.zip includes 16 files: lib0.zip – libf.zip, each 34,902 bytes (34.9 kB)</td>
</tr>
<tr>
<td>Each includes 16 files: book 0.zip - book f.zip, each 29,446 bytes (29.4 kB)</td>
</tr>
<tr>
<td>Each includes 16 files: chapter 0.zip - chapter f.zip, each 32,150 bytes (32.1 kB)</td>
</tr>
<tr>
<td>Each includes 16 files: doc 0.zip - doc f.zip, each 164,302 bytes (165.3 kB)</td>
</tr>
<tr>
<td>Each includes 1 file: 0.dll which size is 4,294,967,295 bytes (4.3 GB)</td>
</tr>
<tr>
<td>Includes just 0xAA</td>
</tr>
</tbody>
</table>

It should be noted that there are different detection levels for 42.zip and .zip-files inside it at different levels. The start of the output when using strings commands to 42.zip can be shown in the table below.

Table - Start of output of command ‘string 42.zip’

$ strings 42.zip
**3.2.3.3 Billion laughs (XML bomb)**

The XML-bomb is a small XML document designed to expand to a gigantic size when parsed by an (unprotected) XML-parser.[19] The purpose of this kind of bomb is to crash the web browser by causing the XML parser to run out of memory. It is claimed in [20] that most current browsers are able to detect such recursive expansion and do not parse the booby-trapped XML.

**3.2.3.4 Other examples**

Harmless_4_ZettaByte_ZIP-bomb.zip is a 5.61kB file that becomes more than 4 zetta bytes (ZB) when fully uncompressed. At the moment there are not enough torrent seeders of the file, so it was not tested. The structure of the torrent file is not described and could not be manually tested, but based on the information from the torrent [21], the structure is close to the structure presented in the table below.

<table>
<thead>
<tr>
<th>Table - Structure of Harmless_4_ZettaByte_ZIP-bomb.zip.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Files</strong></td>
</tr>
<tr>
<td>literature</td>
</tr>
<tr>
<td>--- libraries</td>
</tr>
<tr>
<td>--- library</td>
</tr>
<tr>
<td>--- volume</td>
</tr>
<tr>
<td>--- book</td>
</tr>
<tr>
<td>--- chapter</td>
</tr>
<tr>
<td>--- part</td>
</tr>
<tr>
<td>--- section</td>
</tr>
<tr>
<td>--- page</td>
</tr>
<tr>
<td>--- column</td>
</tr>
<tr>
<td>--- .dll</td>
</tr>
</tbody>
</table>
3.2.4 Fork bombs

Fork bombs operate by consuming CPU power, time and memory. Basic implementation of a fork bomb is an infinite loop that launches the same process repeatedly. It is described in [36] that modern Unix systems generally use copy-on-write when forking new processes and because of this, a fork bomb does not saturate such a system’s memory. We tested some of the example fork bombs described in Wikipedia [15]. Investigation machine used in tests was Kubuntu 14.04 in Virtualbox virtual machine with 1 CPU and 8GB of memory.

Content of executed Python script is presented below.

```python
import os
while True:
    os.fork()
```

This code was saved as forkbomb.py. When starting the script in investigation machine with Python, the top command showed hundreds of similar Python processes. Eventually, it became impossible to do anything with the virtual machine. It was possible to stop the Python script with CTRL-C, even though this took several minutes. After this, using the terminal again became possible, so it was possible to try to execute commands from the terminal. The problem is that the result of executing any (tested) command was:

```
bash: fork: Cannot allocate memory
```

When the terminal where the Python script had been running was closed, everything seemed to come back to normal. The result is that during the execution of a fork bomb, it might not be possible to do anything else other than to try to stop the execution. It might not be possible, for example, to close the terminal windows in GUI and try to stop the execution. Forcing it to stop via CTRL-C might take time. The easiest way seems to be to shut down the investigation machine, but in the worst scenario some of already executed tasks have to be done again and time would be lost. Tests with Microsoft Windows batch language examples from [15] in Windows 7 worked also, and forced to restart the machine physically.

3.2.4.1 Attack scenario

Adversary replaces some of the commands with fork bombs. It would be interesting to try a scenario where live memory or disk imaging tool uses such commands. If a fork bomb is run in the investigation machine, it could cause a DoS attack against it, but luckily the investigator would most likely only lose time.

3.2.4.2 Mitigation techniques

As a mitigation technique, at least the publicly known examples of fork bombs should be searched from files. It should be noted that it would protect only against the known ones and ones that haven’t been encrypted, hidden or obfuscated into the files.

In the investigation machine, the amount of processes that can be spawned should be limited. In the most Linux based distributions this can be done by modifying `/etc/security/limits.conf` file. It is possible to limit amount of number of process for every user. To get more information about this configuration, try `man limits.conf` and `man pam_limits` commands.

3.2.5 Conclusions

There are no problems with opening 42.zip with any Linux computer that has enough resources, not due to lack of resources but because manual archive-opening programs do not have a recursive opening mode. The file was not tested with Windows OS.
It should be noted that mitigation cannot be based on file signatures alone, because it is easy and fast to, for example, add new .zip files inside bombs, change or modify files, or change the amount of ‘0xAA’s or zeros in the final compressed files. It is possible to use the ‘strings’ command to get information about strings inside ZIP bombs, and use this information in the detection of commonly used bombs, but it does not help if the ZIP bombs are uniquely crafted. The investigator shouldn’t use outdated versions of web browsers nor any other outdated tools during investigations. For example, modern web browsers have protection against XML bombs, but older ones [22] might not have. Before opening compressed figures presented in [16] it is possible to check the amount of pixels with exiftool. Images having more pixels than a set limit can be isolated before opening them.

3.3 Data hiding

3.3.1 Hiding data within file slack space

In the NTFS file system, all files are allocated blocks of storage of a certain size regardless of the file’s actual size. Current files are usually stored in 4096 byte clusters which are created from 512 byte sectors. For example, a 3000 byte file might be stored in a 4096 byte cluster leaving 1096 bytes of storage unused. The left over space is known as file slack space.[23] Unless the size of the file increases, file slack is not overwritten. And if the file shrinks, old data still residing within the slack space could be retained.

Slacker is an application that provides the capability to insert files inside file slack space on NTFS file systems.[24] The tool is also part of the Metasploit anti-forensic project. The result of use is that someone can hide evidence within the slack space of another file. It is then more difficult to locate evidence. Although most forensic tools are capable of searching slack space, the main problem is to know which files have had their slack space used.[25] A lot of files, especially those associated with the OS and applications, are updated rarely, if ever. Therefore the slack space of those files is a good place to hide the data.

Testing environment:

1. Windows 7 Enterprise x64 SP1, 1 CPU, 8GB, physical machine
2. Windows XP SP3, 1 CPU, 1GB, Virtual machine

Used forensic tools:

1. AccessData FTK Imager 3.2.0.0
2. ExtractUsnJrnl 1.0.0.1
3. LogFileParser 2.0.0.20
4. Mft2Csv 2.0.0.26
5. UsnJrnl2Csv 1.0.0.6
6. Sleuthkit-4.1.3

### Tool syntax:

#### Hiding a file in slack space:

```
slacker.exe -s <file> <path> <levels> <metadata> [password] [-dxi] [-n1-kl-f <xorfile>]
```

- **-s**: store a file in slack space
- **<file>**: file to be hidden
- **<path>**: root directory in which to search for slack space
- **<levels>**: depth of subdirectories to search for slack space
- **<metadata>**: file containing slack space tracking information
- **[password]**: passphrase used to encrypt the metadata file
- **[-dxi]**: dumb. random. or intelligent slack space selection
- **[-n]**: none, random key, or file based data obfuscation
- **<xorfile>**: the file whose contents will be used as the xor key

#### Restoring a file from slack space:

```
slacker.exe -r <metadata> [password] [-o outfile]
```

- **-r**: restore a file from slack space
- **<metadata>**: file containing slack space tracking information
- **[password]**: passphrase used to decrypt the metadata file
- **[-o outfile]**: output file. Else original location is used. No clobber

### 3.3.1.1 Results

Testing scenario:

- Create a text file with short text inside, use `slacker` to hide the content of the file

Slacker executed with the following syntax:

```
slacker -s c:\secret.txt m:\Files 1 c:\test.jpg -d -n
```

**Explanation of example:**

- **-s** means to store a file in slack space.
- **C:\secret.txt** is the file to hide.
- **m:\Files** holds different files used for slack storage space. In this case it is stored within non-system partition.
- **1** is the level or depth of subdirectories in m:\Files to search for slack space.
- **test.jpg** is used for data storage info (understand like metadata of the hidden content).
- **-d** option for dump slack space selection.
• -n option for none data obfuscation.

From the tool output it is visible that the slack space of the Chrysanthemum.jpg file was used in this case to store data from secret.txt file.

### 3.3.1.2 Analysis and possible mitigation techniques

NTFS $LogFile and change journal ($UsnJrnl) files were exported after the slacker application testing. Output from the NTFS change journal related to slacker application activity:

<table>
<thead>
<tr>
<th>FileName</th>
<th>USN</th>
<th>Timestamp</th>
<th>Reason</th>
<th>MFTReference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrysanthemum.jpg</td>
<td>2668920</td>
<td>2015-10-29 11:02:49.182:5210</td>
<td>DATA_EXTEND</td>
<td>2024</td>
</tr>
<tr>
<td>Chrysanthemum.jpg</td>
<td>2669016</td>
<td>2015-10-29 11:02:49.182:5210</td>
<td>DATA_EXTEND+DATA_TRUNCATION</td>
<td>2024</td>
</tr>
<tr>
<td>Chrysanthemum.jpg</td>
<td>2669112</td>
<td>2015-10-29 11:02:49.198:1210</td>
<td>CLOSE+DATA_EXTEND+DATA_TRUNCATION</td>
<td>2024</td>
</tr>
</tbody>
</table>

Reason explanation:

• Data_extend - data was added to the file.
• Data_truncation - the data in the file was truncated.

Most forensic tools are able to read data from file slack space and in this case, Access Data FTK Imager was used to check data stored in output file slack.

No obfuscation methods were used, and hidden content is readable in clear text form. The same applies also for data storage information as no password was used to obfuscate metadata.
The worse scenario for a forensic analyst occurs if obfuscation is used. The syntax of slacker looks a little different:

```
slacker -s c:\secret.txt m:\Files 1 c:\test.jpg password -x -k
```

Where:
- `password` is passphrase to encrypt the metadata file.
- `-x` option for random slack space selection.
- `-k` option for random key data obfuscation.

Slack space of the output file does not contain clear text information as visible below.
And also password was used to obfuscate metadata therefore also this information is not comprehensible for investigator.

One fact that is noticeable here is the change of the STANDARD_INFORMATION attributes timestamps. They are changed, the new value for M, A, and E timestamps is time when data was added to file slack space.
This change can be easily seen in timeline and if an investigator also has evidence of the tool execution of (from prefetch, UserAssist, or other artefacts), then it can direct analysis to specific file where some data might have been hidden.

Similar to detecting different anti-forensic technique, analysis of the memory can reveal some remnants of any concealment method. It is possible to find proof of use command line tool with volatility plugins cmdscan or consoles. But be aware that this evidence is volatile and usually disappear relatively fast.

In general, analysis of hidden data within the NTFS file system consists of two steps.[26] The first is to find out whether there is hidden data. This can be done by looking for some artefacts that could confirm use of slacker or another tool. The second step is to try to recover the hidden data. It is hard to recover them, because hidden data is usually stored within the file system without any structure or metadata. As it is covered in the previous part, slacker offers certain forms of the output data obfuscation as well as random slack data selection. That makes recovery really challenging or impossible. Forensic tools do not check for hidden data in the NTFS file system except tools that check for alternate data stream (ADS viewer or AltenateStream viewer). And to perform analysis ‘manually’ by looking for suspicious content of slack space is time consuming.

3.3.2 Conclusion

Because of the security features in OS (mentioned also in Timestamp manipulation chapter), slacker does not work correctly from Windows version 6 (Windows 7 and higher + newer versions of the Windows server). It means that it is not possible to hide data within system partition and there is also an issue to recover hidden data from the disk which stores system partition. But still can be used on non-system drives. Slacker works perfectly for Windows XP or Windows Server 2003. Another fact that acts against this tool is that most antivirus products recognise this program as a malicious application.

3.4 Usbkill

As described in [27] usbkill is an anti-forensic kill-switch that shuts down the computer if it notices changes on computer’s USB ports. It is written in Python. It could prevent certain USB related attacks and acquisition.
It should be noted that the tool can be used for good and bad; the person running the script might not be an adversary but an administrator trying to prevent illegal physical access to it.

### 3.4.1 Attack scenario

Three reasons are listed on the tool’s webpage [27]:

- To prevent use of tools that prevent screensavers or sleep mode activating. These might be used by thieves or by the police.
- To prevent retrieving documents from the machine to a USB stick or installation of software via USB.
- To improve the security of the server that is using full disk encryption.

### 3.4.2 Mitigation

To detect the running process, one can investigate running Python processes. It is possible that the adversary will not have changed the name of the script, but this is unlikely. If there are unknown Python processes running, one could examine the script in the particular location. The script is open source, so it is possible to try to discover source files from the storage medias, or strings in source code from files in the storage media, from binary files, or from memory.

### 3.5 Live Linux Distributions

The idea of having removable media for storing operating systems is not new. Since operating systems like MS-DOS in the very early introduction of personal computers, system files were loaded into memory mostly from floppy disks. With the introduction of compact disks, DVDs and USB devices, those types of operating systems became even more popular. These are Live CDs, also known as LiveDistros, which contain a fully operational operating system image file with a boot loader program used to start a computer system. Most of them are available by open source agreement and according to the agreement that ‘anyone can modify and redistribute the original operating system without asking for permission of retribution from the author’. [28]

Despite the fact that Live Distributions are the preferred tools of the trade in conducting digital forensic investigations (SIFT Workstation, DEFT, Caine Live, etc.), most were designed for security testing activities. Great collections of those tools are distributed with Live CD images and are usually used by IT security professionals to troubleshoot computer systems and networks. Unfortunately, malicious users are also taking advantage of them to perform illegal activities. Perpetrators can boot a copy of Linux on a PC, use that machine on a series of computers, turn it off, and walk away without leaving any evidence. They can also run a Live CD on a physical victim machine to modify system files or steal information. A lot of Linux Live Distributions have tools that allow users to dump SAM hashes without leaving any traces. The Windows operating system stores password information on a local hard drive in a system file called System Accounts Manager, also known as a SAM file. Since it is a system file, Windows does not allow a user to access it. Even if we copy that file to a remote location it will be very difficult to decode it because the operating system uses a proprietary encryption utility called a system key. Using distributions like Backtrack or Kali, an attacker can easily obtain this information without leaving any evidence since all the operations are performed in RAM. Any other case such as accessing those files from the local Windows OS will add information about such action into the system log file.

In this sub-chapter, we will identify which activities done from Live Distributions leave traces on the local hard drive, and how to detect them.
3.5.1 Testing environment and the scenario

For testing purposes, we will use Windows 7 Ultimate x86 SP1 operating system with 2 CPU, 2 GB of RAM memory run under VMware Workstation 11.1. Our virtual machine will have only one 10 GB virtual hard drive configured as an independent, persistent, single file with all allocated disk space on a local hard drive. This will give us a configuration that looks as close as possible to the real working machine.

In the scenario, we will use as a Live CD two Linux distributions – Ubuntu 14.04.2 x86 and Puppy Linux 7.0.3 x64. The former is one of the most popular variations of the Linux operating system. The latter is meant to be one of the fastest and stealthiest Linux operating systems distributed for free. They will be tested in three different scenarios:

- First test – boot from Live Distribution on the virtual Windows 7 machine, navigate into two websites www.cnn.com and www.distrowatch.com using the built in web browser (standard mode), copy random data from the website into a newly created text document stored on Live Distribution’s desktop.
- Second test – boot from Live Distribution on the virtual Windows 7 machine, mount Windows’ disk drive (read only mode), navigate to a user’s desktop folder and access secret.txt file, copy the content of the file into a newly created text document on Live Distribution’s desktop.
- Third test – boot from Live Distribution on the virtual Windows 7 machine, mount Windows’ disk drive (read/write mode), navigate to a user’s desktop folder and change the content of the secret.txt file.

Before execution, we created the MD5 sum of the W7-flat.vmdk file which is the representation of a local Windows 7 hard drive. For this purpose we used: WinMD5Free v1.20 made by Liang Ren.

MD5 sum - W7-flat.vmdk: 42bf8afedc94a88bf77f990b5d8047a4

3.5.1.1 Results and possible mitigation techniques.

3.5.1.1.1 First scenario

The first test reveals only one thing – whether Live Distributions leave any traces of evidence on the hard drive. We will not mount the hard drive and we will not touch the existing operating system. We will use a computer just to browse the internet and store information – according to the developers – in the RAM memory. For this purpose, under the Live CD operating system we will open the default web browser, navigate to two designated websites, copy random information from them, and store it on Live Distribution’s desktop in a text document. After that, we will create the MD5 sum from the tested hard drive. This will give as a full picture whether the Live CD requires the user to at some point to touch the hard drive (either allocated or unallocated space).

Hard drive before the test:

MD5 sum - W7-flat.vmdk: 42bf8afedc94a88bf77f990b5d8047a4

Hard drive after the test (Ubuntu 14.04.4):

MD5 sum - W7-flat.vmdk: 42bf8afedc94a88bf77f990b5d8047a4

Hard drive after the test (Puppy Linux 7.0.3):

MD5 sum - W7-flat.vmdk: 42bf8afedc94a88bf77f990b5d8047a4

All MD5 sums are exactly the same. As we can see, there is no way to detect the use of the Live Distribution from the hard drive perspective. If we shut down the machine after the malicious activity (under Ubuntu or
Puppy Linux) then we will not leave any traces in the RAM memory as well. The only possibility to restrict that process is to set additional security rules in BIOS, especially with the new features that come with UEFI. This is the only line of defence against unattended use of Live Distributions.

### 3.5.1.1.2 Second scenario.

The second test is focused on stealing information from a locally stored text document. Again we will boot up the machine using the two Live Distributions, mount the hard drive and navigate to the user’s Windows 7 desktop folder. Then we will open secret.txt file and copy the content to a newly created text document stored on Live Distro’s desktop. By default Ubuntu distributions mount all local hard drives using the following parameters: rw, nosuid, nodev, allow_other, default_permissions, blksize=4096 which means it will allow potential attackers to write in it as well. For the scenario we do not want to do that as it can leave potential evidence on the hard drive, therefore we will use read-only parameter (e.g. `sudo mount -o ro /dev/sda2 /media/windows`).

Hard drive before the test:

MD5 sum - W7-flat.vmdk: 42bf8afedc94a88bf77f990b5d8047a4

Hard drive after the test (Ubuntu 14.04.4):

MD5 sum - W7-flat.vmdk: 42bf8afedc94a88bf77f990b5d8047a4

Hard drive after the test (Puppy Linux 7.0.3):

MD5 sum - W7-flat.vmdk: 42bf8afedc94a88bf77f990b5d8047a4

Again, we can see that mounting the partition in read-only mode does not leave any traces on the hard drive, which is understandable considering our previous scenario. An interesting part is that we created the MD5 sum from the whole disk, and we can see that if we use Live Distributions just for browsing the files or copying them to a remote location it will not leave any traces at all, even in unallocated space. All the action takes place in RAM memory. As we advised before, the only solution would be enforcing a restrictive BIOS security policy to not allow to boot up from random storage media.

### 3.5.1.1.3 Third scenario.

In our last test we will mount the hard drive and modify the secret.txt file stored in a user’s desktop folder. In this case we can already assume that the MD5 sum will change but the question remains whether or not we are able to detect any other attacker’s activity. We are modifying only a text file but this scenario can also apply to mentioned in introduction SAM file or any other system files stored on the hard drive.

Hard drive before the test:

MD5 sum - W7-flat.vmdk: 42bf8afedc94a88bf77f990b5d8047a4

Hard drive after the test (Ubuntu 14.04.4):

MD5 sum - W7-flat.vmdk: 1714f0435fbb069e7c20c7ff518212fc

Hard drive after the test (Puppy Linux 7.0.3):

MD5 sum - W7-flat.vmdk: 763ba0f2884e3d9293fcee85589d7c8b

As we already assumed, the MD5 sum is different for each distribution. Now, in order to detect what happened we will focus only on allocated space on the hard drive modified under the Ubuntu Linux distribution. To examine it we will use AccessData FTK Imager 3.4.0.1.
The hard drive’s structure looks like:

We have two partitions. The first is 100MB and a typical Windows system partition created for booting purposes. The other one is the main storage for all system and user files. In order to compare them with the ones before the test we will create hashes from all the files on a specific partition and export them to a csv document. Then we will compare csv documents to observe differences. For that purpose we used Beyond Compare 4.0.7.

The first partition has 75 files in both before and after test hard drives and no differences whatsoever.

The second partition has 51052 files in both before and after test hard drives and there are ten differences between them shown in the table below.

<table>
<thead>
<tr>
<th>W7-flat.vmdk reference image</th>
<th>MD5 sum</th>
<th>W7-flat.vmdk – after test – Ubuntu</th>
<th>MD5 sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>[root]$I30</td>
<td>ef0c1c2615639a75230</td>
<td>[root]$I30</td>
<td>bbf22bb2256a3d785</td>
</tr>
<tr>
<td>[root]\Users$I30</td>
<td>ef0f02b0343d3f8f46f</td>
<td>[root]\Users$I30</td>
<td>6f0637f64a6392a5e7</td>
</tr>
<tr>
<td>[root]\Users\gibson$I30</td>
<td>7f757f7ca32a9f4e40</td>
<td>[root]\Users\gibson$I30</td>
<td>167cbb744547c86db3</td>
</tr>
<tr>
<td>[root]\Users\gibson\AppData\Roaming\Microsoft\Windows\$I30</td>
<td>89526a782e9143381</td>
<td>[root]\Users\gibson\AppData\Roaming\Microsoft\Windows\$I30</td>
<td>84bfa0ae9cd9c1266</td>
</tr>
<tr>
<td>File not present!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[root]\Users\gibson\Desktop\secret.txt</td>
<td>bc85ef1f4e3f752b5f5</td>
<td>[root]\Users\gibson\Desktop\secret.txt</td>
<td>5fa922f2d4ca4e</td>
</tr>
<tr>
<td>[root]$Bitmap</td>
<td>10063007863922d21e</td>
<td>[root]$Bitmap</td>
<td>a06b1e7f0588cc19</td>
</tr>
<tr>
<td>[root]$MFT</td>
<td>d8f54a1fca573d6e8835</td>
<td>[root]$MFT</td>
<td>7e71f3396b6d5e574</td>
</tr>
<tr>
<td>[root]\Windows\ServiceProfile\Local\LastAlive1.dat</td>
<td>69ed9bdca7e163777c</td>
<td>[root]\Windows\ServiceProfile\Local\LastAlive1.dat</td>
<td>c99a74c05571a433d1</td>
</tr>
<tr>
<td>[unallocated space]</td>
<td>75a1e3aee1ad87875d2</td>
<td>File not present!</td>
<td>2f951d6c6398</td>
</tr>
</tbody>
</table>

We can see that the main difference is between $I30 files and that they are related to the Windows NTFS Index Attribute feature. The basic concept behind indexes is that there is a data structure that is sorted in a specific tree. The tree is made up of two nodes working together – the $Index_Root and $Index_Allocation constructs. $Index_Root is the root of the tree and contains information about the size of the entries and how to sort them. If there are only a few entries, then they will be listed in this attribute. If there are more than a few, then $Index_Root is just the index header and a second attribute, $Index_Allocation, is used to store the actual index entries. The $I30 files contain filenames, file size, creation time, modification time, MFT change time and access time.

Let's see what are the differences between those files in our scenario. For parsing information from index files we will use INDXParse.py python script from github (https://github.com/williballenthin/INDXParse). In the tables below we will show only those parameters that have changed.

First location – [root]\$I30
<table>
<thead>
<tr>
<th>Second location - <code>\[root\]\Users\$I30</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File name</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Third location - <code>\[root\]\Users\gib5on\$I30</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File name</strong></td>
</tr>
<tr>
<td>DOWNLO~1</td>
</tr>
<tr>
<td>ntuser.dat.LOG1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fourth location - <code>\[root\]\Users\gib5on\AppData\Roaming\Microsoft\Windows\$I30</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File name</strong></td>
</tr>
<tr>
<td>Printer</td>
</tr>
<tr>
<td>Templates</td>
</tr>
<tr>
<td>TEMPLA~1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fifth location - <code>\[root\]\Users\gib5on\Desktop\$I30</code></th>
</tr>
</thead>
</table>

Since the file is not present in the reference image, we will list the one was created after scenario.

<table>
<thead>
<tr>
<th>FILENAME, PHYSICAL SIZE, LOGICAL SIZE, MODIFIED TIME, ACCESS TIME, CHANGED TIME, CREATED TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret.txt~ (slack at 0x1a0), 40, 35, 2015-08-31 11:56:08.975487, 2015-08-31 11:56:08.975487, 2015-08-31 11:56:08.975487, 2015-08-31 11:56:08.975487</td>
</tr>
</tbody>
</table>

35
As we can see from index $I30 we can recreate almost the entire attacker’s activity around Windows partitions. We can tell which folders were opened and have evidence of his or her presence in the timestamps. In our case it was easier because we had the reference disk that was used to compare those partitions. In real life, you probably will not have that knowledge but we can always use event logs and check the timeframes when the operating system was up and running. Everything that does not fit into those timeframes should be investigated.

Let’s also check the differences between $MFT files. For that we will use mft2csv tool from google code: https://code.google.com/p/mft2csv/wiki/mft2csv.

Besides the changes of the timestamps mentioned before, there are two differences between a reference and scenario hard drive.

First: in one the file name has been changed – from secret.txt to secret~.txt. This happened because in the scenario we used LibreOffice to modify the .txt file, and as a result the previous version stayed in the same place under a different name; the timestamps did not change.

Second: there is a new record in $MFT table from the image created after the test. It is a new file - secret.txt with new timestamps dated around 2015-09-01 13:39:02 which pinpoints exact time when the file was modified under Ubuntu Live Distribution.

Now, let’s take a look at the secret.txt itself. As we mentioned before, the document was modified with LibreOffice under Ubuntu. In the result of that, the original document was renamed and new one created. The interesting thing about the new document is that the owner name is Administrators (document placed in /Users/gib5on/Desktop/ folder) and NTFS access control has only one group of users – Everyone (SID: S-1-1-0, full access). If you check other documents in /Users/gib5on/ folder you will find that all of them have one owner – gib5on and in NTFS access control there is information regarding at least three users/groups : gib5on, Administrators and SYSTEM (SID: S-1-5-18). Out of that we can suspect that the new secret.txt was not created under a Windows operating system.

3.5.1.2 Key remarks.

Those three tests have proven that if the attacker wants to steal information (read-only mount mode) from the machine using any Linux live distribution there will be no evidence of that on the hard drive. We could also try to search for the evidence in the memory but if the perpetrator is smart enough to power off the machine then remaining hints about his/her activity will be gone as well.

The third scenario showed us a huge weapon to fight not only against timestamps modification but also detection of suspicious activity (either from Live Distribution or unauthorised accounts). This weapon are $I30 index files. Since, they are not $STANDARD_INFORMATION timestamps but $FILE_NAME attribute timestamps, they will be updated more often and in more arbitrary set of circumstances. Moreover, they tend to mirror those in $STANDARD_INFORMATION therefore even if the original file no longer exists, we might still be able to recover its name and original timestamps.

We can imagine a situation where a large number of files were moved to the Recycle Bin, even if at some point it was emptied and most of the files were reallocated. We can still search for additional hints in the $I30 file (bear in mind that the names will be changed according to Recycle Bin schema). Later on, just by analyzing MFT change times of the $I30 entries, it will be possible to learn when the files were moved to the Recycle Bin and recover a list of ‘recycled’ file types using their file extensions. The same approach can be used to malware related incidents.
The only and the most known mitigation technique against Live Distributions is a specific set of rules on BIOS/UEFI level prohibiting booting from external media.

### 3.6 Memory anti-forensics

Numerous anti-forensic techniques and tools can be used to interfere with evidence and CFT. However, memory-based anti-forensic techniques are of particular interest because of their advanced manipulation capabilities for obtaining digital evidence, overall effectiveness, and attacking computer forensic tools. These methods are mainly performed in volatile memory using hiding and advanced data alteration techniques. For these reasons memory-based anti-forensic techniques are considered to be unbeatable. This section aims to present some of the current anti-forensic approaches and in particular reports on memory-based anti-forensic tools and techniques.

#### 3.6.1 ADD – attention deficit disorder

Source: [https://code.google.com/p/attention-deficit-disorder/](https://code.google.com/p/attention-deficit-disorder/)

The primary goal of this program is to confuse the analysis process. Most of the time analysts are focused on the output of the tool. That’s why all one needs to do is to place additional artefacts in the memory not related to investigating an incident and increase the noise to signal ratio. This can cause reduced trust in the forensic software or the captured memory image itself. Additionally, more artefacts in the memory make the analysis process more time consuming.

The ADD tool mainly focuses on three aspects:

- Adding fake filenames strings in the memory.
- Creating terminated processes with additional attributes like PID (Process ID), PPID (Parent Process ID) and date of creation.
- Create TCP connections with artefacts in memory using attributes like local IP address, local port, destination IP address, destination port and the state of the connection.

C:\A\P\ADD>ADD_User_Console.exe

Usage:

```
/filo filename
/proc procname pid ppid createtime createfinel exitfinel exitfinel
/tcpCon localIP localPort remoteIP remotePort state
```

Note that IP addresses is expressed as decimal, not dotted quad.
State should probably usually be 4 (connected)

**ADD tool – general use**

#### 3.6.1.1 Testing environment

Operating systems:

- Windows 7 Ultimate x86 SP1, 2 CPU, 2GB, VMware Workstation 11.1 VM
- Windows Server 2008 x86 SP2, 2 CPU, 2GB, VMware Workstation 11.1 VM
- Windows Server 2012 R2, 2 CPU, 4GB, VMware Workstation 11.1 VM

Memory image acquisition tools:

- AccessData FTK Imager 3.1.1.8 – portable edition
- MoonSols Windows Memory DumpIt 1.3.2.20110401
- Win32dd/Win64dd 1.3.1.20100417

Memory image analysis tool: Volatility Framework 2.4.
3.6.1.2 Testing scenario

- inject two fake processes into memory using the ADD tool – fake101.exe (PID: 3333, PPID: 500), fake102.exe (PID:3444, PPID: 500)
- inject two fake filenames into memory using the ADD tool – fake101.exe, fake102.exe
- inject two fake connections from 192.168.1.66:46785 to 123.123.123.88 and 192.168.1.66:46786 to 123.123.123.124:88

ADD tool – testing scenario

3.6.1.3 Installation

The source code is available at code.google.com. For testing purposes, we used a compiled version of executable located in the ‘Release’ folder (ADD_User_Console.exe). In order to get ADD tool working, it is necessary to load a driver add.sys provided from separate zip file – add-file.zip is available on the same website. OSRLoader (v. 3.0) was used to load a driver into an operating system. The executable and the driver used for the test were already compiled, but it is possible to download their source code and modify it according to one’s needs.

3.6.1.4 Volatility Framework 2.4 output

The main goal for the testing scenario was to observe whether the tool works which depends on the memory imaging tool and how the processes and files are being planted in the memory. We tested three different operating systems. Given that in order to run ADD tool it is necessary to load specially crafted driver (32bit version), we were not able to run it on Windows 2012 R2. After running scenario on Windows 7 and Windows Server 2008 we took three memory images per operating system using three different imaging tools (see the testing environment).

pstree – list all active processes (before using ADD tool):

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>PPid</th>
<th>Thds</th>
<th>Hnds</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x85e95030:wininit.exe</td>
<td>404</td>
<td>304</td>
<td>7</td>
<td>93</td>
<td>2015-08-17 06:40:19 UTC+0000</td>
</tr>
<tr>
<td>. 0x85f97d28:lsm.exe</td>
<td>516</td>
<td>404</td>
<td>12</td>
<td>156</td>
<td>2015-08-17 06:40:19 UTC+0000</td>
</tr>
<tr>
<td>.. 0x85f63030:services.exe</td>
<td>500</td>
<td>404</td>
<td>21</td>
<td>253</td>
<td>2015-08-17 06:40:19 UTC+0000</td>
</tr>
<tr>
<td>.. 0x870481e0:taskhost.exe</td>
<td>1536</td>
<td>500</td>
<td>12</td>
<td>227</td>
<td>2015-08-17 06:40:20 UTC+0000</td>
</tr>
<tr>
<td>.. 0x85f76830:svchost.exe</td>
<td>2408</td>
<td>500</td>
<td>6</td>
<td>123</td>
<td>2015-08-17 06:40:20 UTC+0000</td>
</tr>
<tr>
<td>.. 0x870481e0:taskhost.exe</td>
<td>780</td>
<td>500</td>
<td>22</td>
<td>436</td>
<td>2015-08-17 06:40:19 UTC+0000</td>
</tr>
<tr>
<td>.. 0x85ec3318:audiogd.exe</td>
<td>964</td>
<td>780</td>
<td>6</td>
<td>133</td>
<td>2015-08-17 06:40:19 UTC+0000</td>
</tr>
</tbody>
</table>
### psscan – list hidden and terminated processes (before using ADD tool)

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Time exited</th>
<th>Name</th>
<th>PID</th>
<th>PPID</th>
<th>PDB</th>
<th>Time created</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000007dcdb458</td>
<td>3333</td>
<td>fake101.exe</td>
<td>500</td>
<td>0x12345680</td>
<td>2015-07-23 23:36:01</td>
<td></td>
</tr>
<tr>
<td>2015-07-23 23:43:11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000007de4e848</td>
<td>3444</td>
<td>fake102.exe</td>
<td>500</td>
<td>0x12345680</td>
<td>2015-07-23 23:36:11</td>
<td></td>
</tr>
<tr>
<td>2015-07-23 23:58:20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### psscan – list hidden and terminated processes (after using ADD tool)

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Time exited</th>
<th>Name</th>
<th>PID</th>
<th>PPID</th>
<th>PDB</th>
<th>Time created</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000007dcdb458</td>
<td>3333</td>
<td>fake101.exe</td>
<td>500</td>
<td>0x12345680</td>
<td>2015-07-23 23:36:01</td>
<td></td>
</tr>
<tr>
<td>2015-07-23 23:43:11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x00000007de4e848</td>
<td>3444</td>
<td>fake102.exe</td>
<td>500</td>
<td>0x12345680</td>
<td>2015-07-23 23:36:11</td>
<td></td>
</tr>
<tr>
<td>2015-07-23 23:58:20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### psxview – cross reference processes with various lists (before using ADD tool):

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Name</th>
<th>PID</th>
<th>ExitTime</th>
<th>pplist psscan thrdproc papcict csrss session deskthrd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7df7bc00</td>
<td>vmtoolsd.exe</td>
<td>352</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7df5a340</td>
<td>WmiPrvSE.exe</td>
<td>200</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7de709a8</td>
<td>vmtoolsd.exe</td>
<td>1616</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e307308</td>
<td>SearchIndexer</td>
<td>2528</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e0f7510</td>
<td>avchost.exe</td>
<td>992</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e0f73d8</td>
<td>avchost.exe</td>
<td>816</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e0c3318</td>
<td>audiody.dll</td>
<td>964</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e253030</td>
<td>TAutoConnSvc</td>
<td>628</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e7c132c0</td>
<td>mdsvc.exe</td>
<td>2192</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e3f4d50</td>
<td>conhost.exe</td>
<td>500</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e1b2d28</td>
<td>avchost.exe</td>
<td>1304</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e133250</td>
<td>SearchProtocol</td>
<td>2616</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e27030</td>
<td>avchost.exe</td>
<td>1432</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e9baa8</td>
<td>dwm.exe</td>
<td>1664</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e372030</td>
<td>appsvc.exe</td>
<td>2844</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f097d28</td>
<td>svchost.exe</td>
<td>612</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f163030</td>
<td>services.exe</td>
<td>500</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f15e508</td>
<td>svchost.exe</td>
<td>856</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f23fb88</td>
<td>taskeng.exe</td>
<td>3536</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f10b6c0</td>
<td>avchost.exe</td>
<td>884</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f102540</td>
<td>lsass.exe</td>
<td>508</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f1e481e0</td>
<td>taskhost.exe</td>
<td>1536</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7fde0e0</td>
<td>Gwx.exe</td>
<td>2812</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7de4a9c8</td>
<td>avchost.exe</td>
<td>956</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e3394b0</td>
<td>SearchFilterHo</td>
<td>2636</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e8ab2d2</td>
<td>explorer.exe</td>
<td>1724</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e284850</td>
<td>TAutoConnect</td>
<td>2304</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f0ff030</td>
<td>winlogon.exe</td>
<td>460</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7fe6d2d8</td>
<td>d1host.exe</td>
<td>2084</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e264300</td>
<td>conhost.exe</td>
<td>2316</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f864a0</td>
<td>WmiPrvSE.exe</td>
<td>768</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e191030</td>
<td>spoolsv.exe</td>
<td>1256</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f0e4e0</td>
<td>avchost.exe</td>
<td>688</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e291b90</td>
<td>VSSVC.exe</td>
<td>2408</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7e131d28</td>
<td>avchost.exe</td>
<td>1116</td>
<td>True False True True True True</td>
<td></td>
</tr>
<tr>
<td>0x7f166830</td>
<td>svchost.exe</td>
<td>780</td>
<td>True False True True True True</td>
<td></td>
</tr>
</tbody>
</table>
### psxview – cross reference processes with various lists (after using ADD tool):

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Name</th>
<th>PID</th>
<th>plist</th>
<th>pslist</th>
<th>thrdproc</th>
<th>pspcid</th>
<th>csrss</th>
<th>session</th>
<th>deskthrd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7f2f59e8</td>
<td>DumpIt.exe</td>
<td>3488</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e253d28</td>
<td>dllhost.exe</td>
<td>1968</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7f197d28</td>
<td>lsass.exe</td>
<td>516</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7f337200</td>
<td>smss.exe</td>
<td>232</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>0x7f179d28</td>
<td>csrss.exe</td>
<td>316</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7f0a0b30</td>
<td>csrss.exe</td>
<td>396</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x77fc86f0</td>
<td>System</td>
<td>4</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>0x7e011030</td>
<td>dllhost.exe</td>
<td>3416</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

2015-07-17 06:40:55 UTC+0000

### netscan – scan for connections and sockets (before using ADD tool):

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Proto</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>State</th>
<th>Pid</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7df58cf8</td>
<td>TCPv4</td>
<td>0.0.0.0:445</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>4</td>
<td>System</td>
</tr>
<tr>
<td>0x7e1be6e0</td>
<td>TCPv4</td>
<td>0.0.0.0:49155</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>500</td>
<td>services.exe</td>
</tr>
<tr>
<td>0x7e011030</td>
<td>TCPv4</td>
<td>192.168.167.133:139</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>4</td>
<td>System</td>
</tr>
</tbody>
</table>

2015-07-17 06:48:50 UTC+0000

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Proto</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>State</th>
<th>Pid</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7dd44030</td>
<td>TCPv4</td>
<td>0.0.0.0:445</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>4</td>
<td>System</td>
</tr>
<tr>
<td>0x7dcdb458</td>
<td>TCPv4</td>
<td>0.0.0.0:49155</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>500</td>
<td>services.exe</td>
</tr>
<tr>
<td>0x7df58cf8</td>
<td>TCPv4</td>
<td>192.168.167.133:139</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>4</td>
<td>System</td>
</tr>
</tbody>
</table>

2015-07-17 23:58:20 UTC+0000
Although ADD tool shows output under Windows Server 2008 x86, we haven’t seen any injected processes and files. From the developer’s website we can see that ADD was prepared mainly for Windows 7, and that is why in its current state it will not work under any other Microsoft operating system. Giving the fact that the project has open source code it is possible to see in the future other versions built for different operating systems.

3.6.1.5 Analysis of the results and possible mitigation techniques.

All images present same output under Volatility. Out of that, we can presume that different memory acquisition tools are equally affected by ADD.

Results:

- **psscan** – shows two closed down processes. The PDB (page directory base) is the same on all two fake processes. This is a static signature in the released build of ADD, and any attacker could change this; although, it is likely script kiddies will not. In any case, each process should have a unique PDB. If we want to signature the default build of ADD, this is currently our best bet. The PDB of 0x12345680 is very unlikely to be legitimate (and very likely to belong to ADD). Also, we should check the time stamps. They shouldn’t be before the system was restarted.

- **psxview** – enumerate processes using different techniques. If you attempt to hide processes, you will have to evade detection some different ways. Looking at the output, the only way we can see that there were those suspicious looking processes running (at least at one point in time) was by scanning for EPROCESS structures.

- **netscan** – gives two suspicious connections to 123.123.123.123, and from the command output we cannot see much difference between injected and legitimate ones. The only thing that stands out is PID – 0. This is also the static signature in the released build of ADD, and is our best bet is to mitigate injected communication artefacts.

Let’s focus on hidden/terminated processes and investigate them.
First, we will try command – dlllist – list of loaded libraries by suspicious process fake101.exe.

```
vol.py -f w7_add_after.raw --profile=Win7SP1x86 dlllist -p 3333
```

Volatility Foundation Volatility Framework 2.4
ERROR : volatility.plugins.taskmods: Cannot find PID 3333. If its terminated or unlinked, use psscan and then supply --offset=OFFSET

The volatility gives us an error, because with terminated or unlinked processes we have to supply the offset instead.

```
vol.py -f w7_add_after.raw --profile=Win7SP1x86 dlllist --offset=0x00000007dcdb458
```

Volatility Foundation Volatility Framework 2.4
*************************************************** *********************
- pid: -----
Unable to read PEB for task.
There are no libraries related to that process. This may indicate that the process was injected.

Second – the handles command lists the handles for a process called fake101.exe.

```
vol.py -f w7_add_after.raw --profile=Win7SP1x86 handles --offset=0x00000007dcdb458
```

Volatility Foundation Volatility Framework 2.4
Offset(V)  Pid  Handle  Access Type           Details
---------- ------ ---------- ---------- ----------- --------------- -------
There are no libraries related to that process. This may indicate that the process was injected.

Second – the handles command lists the handles for a process called fake101.exe.

```
vol.py -f w7_add_after.raw --profile=Win7SP1x86 handles --offset=0x00000007dcdb458
```

Volatility Foundation Volatility Framework 2.4
Offset(V)  Pid  Handle  Access Type           Details
---------- ------ ---------- ---------- ----------- --------------- -------
There are no handles related to that process.

Next we will try to obtain processes by dumping to executable sample using plugin procdump.

```
vol.py -f w7_add_after.raw --profile=Win7SP1x86 procdump --offset=0x00000007dcdb458 --dump-dir=/cases/add/
```

Volatility Foundation Volatility Framework 2.4
Process(V) ImageBase Name        Result
---------- ---------- -------------------- -----
---------- ------- -----------------------
Error: Cannot acquire process AS

We can also try to dump the memory section into a file using plugin memdump.

```
vol.py -f w7_add_after.raw --profile=Win7SP1x86 memdump --offset=0x00000007dcdb458 --dump-dir=/cases/add/
```

Volatility Foundation Volatility Framework 2.4
We are unable to dump suspicious processes. This along with no handles and related dlls may indicate that memory might have been tampered with an anti-forensic tool.
Knowing that the ADD tool needs a special driver to inject artefacts into memory, let’s move on to loaded, unloaded and unlinked drivers using `modscan`.

```plaintext
vol.py -f w7_add_after.raw --profile=Win7SP1x86 modscan
```

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Name</th>
<th>Base</th>
<th>Size</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000007dd14ec8</td>
<td>asyncmac.sys</td>
<td>0xa0925000</td>
<td>0x9000</td>
<td>\SystemRoot\system32\DRIVERS\asyncmac.sys</td>
</tr>
<tr>
<td>0x000000007de02d90</td>
<td>mrxsmb.sys</td>
<td>0x97cc0000</td>
<td>0x23000</td>
<td>\SystemRoot\system32\DRIVERS\mrxsmb.sys</td>
</tr>
<tr>
<td>0x000000007de08188</td>
<td>mrxsmb20.sys</td>
<td>0x97d1f000</td>
<td>0x1c000</td>
<td>\SystemRoot\system32\DRIVERS\mrxsmb20.sys</td>
</tr>
<tr>
<td>0x000000007deb01e8</td>
<td>srv.sys</td>
<td>0xa0869000</td>
<td>0x52000</td>
<td>\SystemRoot\System32\DRIVERS\srv.sys</td>
</tr>
<tr>
<td>0x007f0fb688</td>
<td>DumpIt.sys</td>
<td>0xa0936000</td>
<td>0x0000</td>
<td>??\C:\Windows\system32\Drivers\DumpIt.sys</td>
</tr>
<tr>
<td>0x000000007f105330</td>
<td>drmk.sys</td>
<td>0x9368b000</td>
<td>0x19000</td>
<td>\SystemRoot\System32\drivers\drmk.sys</td>
</tr>
<tr>
<td>0x000000007f2517f0</td>
<td>crashdmp.sys</td>
<td>0x936a4000</td>
<td>0x0000</td>
<td>\SystemRoot\System32\Drivers\crashdmp.sys</td>
</tr>
<tr>
<td>0x000000007fa62e38</td>
<td>spsys.sys</td>
<td>0xa08bb000</td>
<td>0x6a00</td>
<td>\SystemRoot\system32\drivers\spsys.sys</td>
</tr>
<tr>
<td>0x000000007fa65a48</td>
<td>add.sys</td>
<td>0xa092e000</td>
<td>0x8000</td>
<td>??\C:\AF\ADD\add.sys</td>
</tr>
</tbody>
</table>

Two locations seem odd – the first location DumpIt.sys indicates an imaging tool, but the second one, add.sys, gives us some additional information about the tool that was used.

Let’s scan services using a plugin – `svcscan`.

```plaintext
vol.py -f w7_add_after.raw --profile=Win7SP1x86 svcscan
```

<table>
<thead>
<tr>
<th>Offset: 0x7c09c0</th>
<th>Order: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start: SERVICE_DEMAND_START</td>
<td></td>
</tr>
<tr>
<td>Process Name: add</td>
<td></td>
</tr>
<tr>
<td>Display Name: add</td>
<td></td>
</tr>
<tr>
<td>Service Type: SERVICE_KERNEL_DRIVER</td>
<td></td>
</tr>
<tr>
<td>Service State: SERVICE_RUNNING</td>
<td></td>
</tr>
<tr>
<td>Binary Path: \Driver\add</td>
<td></td>
</tr>
</tbody>
</table>

We have a suspicious location for a driver – add.sys and a service called add. In our case, we used build version. The attacker can, of course, change it, as the whole project is open-source. In spite of that, the ADD tool requires a custom driver and service under Windows that can be detected.

Now let’s try to recover the command history using `cmdscan`.

```plaintext
*******************************************************************************
CommandProcess: conhost.exe Pid: 2408
CommandHistory: 0x28c9d0 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
Cmd #5 @ 0x53004d: ?I
*******************************************************************************
CommandProcess: conhost.exe Pid: 3504
CommandHistory: 0x3b9a50 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 13 LastAdded: 12 LastDisplayed: 12
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x60
Cmd #0 @ 0x39d2e8: cd / 
Cmd #1 @ 0x3b8550: dir 
Cmd #2 @ 0x39d300: add.exe
Cmd #3 @ 0x39d330: clear 
Cmd #4 @ 0x39d348: clean 
Cmd #5 @ 0x3b8570: c:
Cmd #6 @ 0x3b8580: cls 
Cmd #7 @ 0x3ba668: add.exe /file fake101.exe
Cmd #8 @ 0x3beee90: add.exe /file fake102.exe
Cmd #9 @ 0x3beed0: add.exe /proc fake101.exe 3333 500 1438858636 30459296 1438858636 30459297
Cmd #10 @ 0x3bef70: add.exe /proc fake102.exe 3444 500 1538858636 30459296 1938858636 30459299
```
From the output, we can see several items of interest. Program add.exe injected all fake artefacts used for our scenario, which pretty much explains everything what has happened. Given the fact that we cannot always rely on command line history (can be easily overwritten) let's try to search for those artefacts in strings.

First, we have to create ascii and unicode strings files.

```
srch_strings -a -t d w7_add_after.raw > strings.txt
srch_strings -a -t d -el w7_add_after.raw >> strings.txt
```

Then, let's browse `strings.txt` file for suspicious processes.

```
cat strings.txt | grep fake101.exe
```

Even if we do not have any history from the command line, it's always advisable to check for any suspicious strings in the memory image. This also gives us ideas on how those artefacts got into the analysing machine.

Now when we have a suspicious add.exe file, we can browse it in strings.exe and see if it yields any results.
cat strings.txt | grep add.exe

200262684 add.exe
2011061428 add.exe
2012776948 add.exe
2023762532 add.exe
2107657356 add.exe
2111392020 add.exe
2113330836 add.exe
2140368052 add.exe
1082798 *C:\add.exe
1082824 \??\C:\add.exe
7188502 ;add.exe /file fake101.exe
string pointer = 0x870687c0 string length= 22
FileObj Pool pointer = 0x873c9ee8
C:\>add.exe /file fake102.exe string pointer = 0x86fa34d0
C:\>add.exe /proc fake101.exe
3333 500 1438858636
2000262690 eadd.exe /file fake101.exe
string pointer = 0x870687c0 string length= 22
FileObj Pool pointer = 0x873c9ee8
C:\>add.exe /file fake102.exe
string pointer = 0x86fa34d0
C:\>add.exe /proc fake101.exe
3333 500 1438858636
This output gives us a full picture of what happened. It also gives us a location of the executable that was used to inject the process, file and network related fake artefacts. We can also see that those hints are put in a lot of different places. That is why even if we do not have any results from the cmdscan or consoles plugins, it is still possible to get some results.

Key remarks:

- The best bet is to search for a solution in the strings, but, first of all, we need at least some hints about suspicious processes or drivers.
- Be advised that we will not always get results from strings as it can be randomly overwritten.
- Try to search for static signatures in released build like PDB parameter, if we are dealing with more than one injected processes and lazy attacker we will have the same PDB in all fake processes.
- If the process has no handles, no dlls, and it is impossible to obtain it together with broken timestamps, this may indicate that the memory was tampered with an anti-forensic tool.

In brief, ADD can be a very powerful tool in a favourable environment. If the attacker can enhance it – PDB randomiser, cleaning cmd history, removing PID 0 under TCP connection artefacts and paying attention to timestamps – than analysis of that would be much more difficult and would take a lot more time. It can also make analysis almost impossible if this tool was only used to inject TCP connection artefacts. Given the fact that the build signature of the released file is not recognised by virustotal.com, we can assume it will not be picked up by antivirus software. The primary concern of the attacker is always that he requires full administrative rights, especially the part necessary to inject required driver.

### 3.6.2 Dementia

Source: [https://code.google.com/p/dementia-forensics/](https://code.google.com/p/dementia-forensics/)

Dementia is an anti-forensic toolkit designed for hiding various artefacts inside the memory dump during memory acquisition on Microsoft Windows operating systems. There are two major problems with acquisition tools. First, they are run on machines controlled by the handler, meaning attackers can have kernel-level visibility and control over the system. Second, tools must dump their data on a local or external disk or on a networked machine. By combining these two facts and controlling the process of dump writing, attackers can defeat most live memory acquisition methods forensic experts and incident handlers use.

Let’s take a closer look at how memory acquisition looks like in Windows operating system. First, we need an executable with a user interface that will allow us to get the job done. Along with the executable comes a specific driver which either reads directly from memory device `\Device\PhysicalMemory` or maps physical memory space using function `MmMapIoSpace()`. Then, in order to dump the content of the memory into a single file, the acquisition tool uses Windows API function called `NtWriteFile()`. The result is a file that represents the memory content of the machine at the moment of acquisition.

Now, knowing what the whole process looks like we can start thinking about where the weakest link is in it. Most of the acquisition tools have patterns – specific Windows API `NtWriteFile()` call arguments, known process or driver names and unique `FILE_OBJECT` values or flags. If an anti-forensic tool is able to recognise the pattern
of memory acquisition and controls dumping to a file activity then we can presume that the whole process may or will be compromised. As suspected, Dementia uses this vulnerability.

The tool supports three hiding methods:

- **User-mode hiding** – applications like Mandiant Memoryze or FTK Imager use their drivers as a ‘proxy’ for kernel access. The driver maps physical memory and sends the contents back to the user-mode. At the moment this method is supported only for Mandiant Memoryze. It injects a DLL into Memoryze.exe process, hooks DeviceIoControl() function and clears the contents of the dump.

- **Kernel-mode hiding** is based on hooks. It installs a driver which performs inline hooking of NtWriteFile and NtClose functions. It then waits for memory acquisition software using various heuristic process names or drivers, specific NtWriteFile() arguments or specific FILE_OBJECT values and flags. After detection, it builds a list of addresses and ranges that need to be hidden. When a buffer containing a ‘tagged’ address is being written to a memory dump, the driver sanitises the buffer and removes the artefacts from the dump. Due to additional kernel level protections this method is supported only on 32-bit systems.

- **Kernel-mode hiding based on a file system mini-filter driver.** Basically this method works exactly the same as the previous one, but instead of inline hooking of kernel functions, an additional driver is created which works as a file system mini-filter driver, sanitising the buffer as it is being written to disk.

- **Dementia – general use**

3.6.2.1 **Testing environment**

Operating systems:
- Windows 7 Ultimate x86 SP1, 2 CPU, 2GB, VMware Workstation 11.1 VM
- Windows 7 Ultimate x64 SP1, 2 CPU, 4 GB, VMware Workstation 11.1 VM
- Windows Server 2008 x86 SP2, 2 CPU, 2GB, VMware Workstation 11.1 VM
- Windows Server 2008 x64 SP2, 2 CPU, 4 GB, VMware Workstation 11.1 VM
- Windows Server 2012 R2, 2 CPU, 4GB, VMware Workstation 11.1 VM
Memory image acquisition tools:
- AccessData FTK Imager 3.1.1.8 – portable edition
- MoonSols Windows Memory DumpIt 1.3.2.20110401
- Win32dd/Win64dd 1.3.1.20100417 (all three methods)
- Belkasoft Live RAM Capturer
- Mdd 1.3 – ManTech Physical Memory Dump Utility

Memory image analysis tool - Volatility Framework 2.4

3.6.2.2 Testing scenario
- run ProcessHacker.exe in a separate window
- hide processes – ProcessHacker.exe (PID 3324) and cmd.exe (PID 3856)
- hide processes related to Dementia – Dementia.exe and conhost.exe

3.6.2.3 Installation.
The source code is available at code.google.com. Given the fact that the first method focuses on only one acquisition tool, our test will focus only on kernel-mode hiding methods and try to test as many acquisition tools as possible. For testing purposes, we used a pre-compiled version of the executable. Dementia is an open-source code, it’s possible to download it and modify according to the user’s need which makes detection process even more complicated. Program was run from command line with administrative rights.

3.6.2.4 Volatility Framework 2.4 output
The main goal for the testing scenario was to observe how the tool works and whether it supports the most popular memory image acquisition tools. We tested five different operating systems. The ‘kernel mode hiding based on hooks’ method was used on x86 architecture and kernel-mode hiding based on file system mini-filter method was used on 64 bit systems. Under Windows Server 2012 R2 we experienced system crashes (BSOD) therefore at this moment we can presume that this system in not affected by Dementia’s mini-filter driver. After running the scenario on Windows 7 and Windows Server 2008 both x86 and x64, we tested all imaging tools mentioned in a testing environment. All of those tools showed the same results; therefore, in the analysis we focused on the Windows 7 x86 image captured by Moonsols DumpIt. Win32dd and Win64dd came up with three different mapping methods during acquisition – MmMapIoSpace(), \Device\PhisicalMemory andPFN Mapping.. All of them presented the same output:

\pstr - list all active processes (before using Dementia)

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>PPid</th>
<th>Thds</th>
<th>Hnds</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>explorer.exe</td>
<td>1764</td>
<td>1700</td>
<td>34</td>
<td>880</td>
<td>2015-08-21 11:40:18 UTC+0000</td>
</tr>
<tr>
<td>FTK Imager.exe</td>
<td>3968</td>
<td>1764</td>
<td>20</td>
<td>392</td>
<td>2015-08-21 11:40:58 UTC+0000</td>
</tr>
<tr>
<td>vmtoolsd.exe</td>
<td>1412</td>
<td>1764</td>
<td>8</td>
<td>181</td>
<td>2015-08-21 11:40:19 UTC+0000</td>
</tr>
<tr>
<td>ProcessHacker.exe</td>
<td>2920</td>
<td>1764</td>
<td>9</td>
<td>282</td>
<td>2015-08-21 11:40:39 UTC+0000</td>
</tr>
<tr>
<td>wininit.exe</td>
<td>404</td>
<td>296</td>
<td>7</td>
<td>88</td>
<td>2015-08-21 11:40:16 UTC+0000</td>
</tr>
<tr>
<td>lsass.exe</td>
<td>542</td>
<td>404</td>
<td>11</td>
<td>152</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
</tr>
<tr>
<td>services.exe</td>
<td>516</td>
<td>404</td>
<td>10</td>
<td>610</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
</tr>
<tr>
<td>services.exe</td>
<td>508</td>
<td>404</td>
<td>19</td>
<td>239</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
</tr>
<tr>
<td>Process Name</td>
<td>PID</td>
<td>Ppid</td>
<td>Children</td>
<td>Start Time</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----</td>
<td>------</td>
<td>----------</td>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td>spoolsv.exe</td>
<td>1284</td>
<td>508</td>
<td>18</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>vssvc.exe</td>
<td>2440</td>
<td>508</td>
<td>6</td>
<td>2015-08-21 11:40:21 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>876</td>
<td>508</td>
<td>44</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>taskeng.exe</td>
<td>3116</td>
<td>876</td>
<td>6</td>
<td>2015-08-21 11:40:48 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>784</td>
<td>508</td>
<td>23</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>audioldg.exe</td>
<td>976</td>
<td>784</td>
<td>6</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>vmtoolsd.exe</td>
<td>1552</td>
<td>508</td>
<td>11</td>
<td>2015-08-21 11:40:18 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>TPAutoConnSvc</td>
<td>240</td>
<td>508</td>
<td>10</td>
<td>2015-08-21 11:40:19 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>TPAutoConnect</td>
<td>2224</td>
<td>240</td>
<td>6</td>
<td>2015-08-21 11:40:20 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>1316</td>
<td>508</td>
<td>25</td>
<td>2015-08-21 11:40:18 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>dllhost.exe</td>
<td>636</td>
<td>508</td>
<td>22</td>
<td>2015-08-21 11:40:19 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>1460</td>
<td>508</td>
<td>13</td>
<td>2015-08-21 11:40:18 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>824</td>
<td>508</td>
<td>24</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>dwm.exe</td>
<td>1716</td>
<td>824</td>
<td>7</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>2832</td>
<td>508</td>
<td>7</td>
<td>2015-08-21 11:40:34 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>1004</td>
<td>508</td>
<td>7</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>848</td>
<td>508</td>
<td>19</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>696</td>
<td>508</td>
<td>13</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>taskhost.exe</td>
<td>1636</td>
<td>508</td>
<td>12</td>
<td>2015-08-21 11:40:18 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>2332</td>
<td>508</td>
<td>7</td>
<td>2015-08-21 11:40:20 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>mdm.exe</td>
<td>2152</td>
<td>508</td>
<td>16</td>
<td>2015-08-21 11:40:20 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>620</td>
<td>508</td>
<td>17</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>252</td>
<td>620</td>
<td>8</td>
<td>2015-08-21 11:40:19 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>1136</td>
<td>508</td>
<td>22</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td>372</td>
<td>508</td>
<td>19</td>
<td>2015-08-21 11:40:19 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>SearchIndexer</td>
<td>2540</td>
<td>508</td>
<td>15</td>
<td>2015-08-21 11:40:25 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>SearchFilterHo</td>
<td>2636</td>
<td>2540</td>
<td>5</td>
<td>2015-08-21 11:40:25 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>SearchProtocol</td>
<td>2616</td>
<td>2540</td>
<td>7</td>
<td>2015-08-21 11:40:25 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>csrss.exe</td>
<td>312</td>
<td>296</td>
<td>9</td>
<td>2015-08-21 11:40:12 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>csrss.exe</td>
<td>412</td>
<td>396</td>
<td>10</td>
<td>2015-08-21 11:40:16 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>conhost.exe</td>
<td>2244</td>
<td>412</td>
<td>1</td>
<td>2015-08-21 11:40:20 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>winlogon.exe</td>
<td>468</td>
<td>396</td>
<td>6</td>
<td>2015-08-21 11:40:17 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>GWX.exe</td>
<td>2800</td>
<td>2788</td>
<td>6</td>
<td>2015-08-21 11:40:34 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>4</td>
<td>0</td>
<td>82</td>
<td>2015-08-21 11:40:15 UTC+0000</td>
<td></td>
</tr>
<tr>
<td>smss.exe</td>
<td>2</td>
<td>28</td>
<td>4</td>
<td>2015-08-21 11:40:15 UTC+0000</td>
<td></td>
</tr>
</tbody>
</table>

To show the differences between acquired memory image and real system status, the screen shot below shows Process Hacker output just before image acquisition.
```
pstree – list all active processes (after using Dementia)
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>PPid</th>
<th>Thds</th>
<th>Hnds</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>svchost.exe</td>
<td>832</td>
<td></td>
<td></td>
<td></td>
<td>3.59 MB Host Process</td>
</tr>
<tr>
<td>dwm.exe</td>
<td>1620</td>
<td>0</td>
<td>1.43</td>
<td>0</td>
<td>4.76 MB Desktop Window Manager</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>856</td>
<td>1016</td>
<td>0</td>
<td></td>
<td>11.30 MB Host Process</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>892</td>
<td>1164</td>
<td>0</td>
<td></td>
<td>10.86 MB Host Process</td>
</tr>
<tr>
<td>spoolsv.exe</td>
<td>1304</td>
<td>1344</td>
<td>0</td>
<td></td>
<td>5.54 MB Spool SubSystem App</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>1344</td>
<td>1344</td>
<td>0</td>
<td></td>
<td>9.07 MB Host Process</td>
</tr>
<tr>
<td>taskhost.exe</td>
<td>1484</td>
<td>0</td>
<td>0</td>
<td>1970-01-01 00:00:00 UTC+0000</td>
<td>10.65 MB Host Process</td>
</tr>
<tr>
<td>svchost.exe</td>
<td>1512</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>3.39 MB Host Process</td>
</tr>
<tr>
<td>vmtoolsd.exe</td>
<td>1704</td>
<td>1704</td>
<td>0.02</td>
<td>0</td>
<td>6.68 MB VMware Tools Core Service</td>
</tr>
<tr>
<td>TPAutoConnect.exe</td>
<td>332</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1.54 MB ThinPrint AutoConnect printer</td>
</tr>
<tr>
<td>explorer.exe</td>
<td>1740</td>
<td>9772</td>
<td>0.03</td>
<td>0</td>
<td>24.36 MB Windows Explorer</td>
</tr>
<tr>
<td>vmtoolsd.exe</td>
<td>416</td>
<td>416</td>
<td>0.07</td>
<td>0</td>
<td>6.37 KB/s VMware Tools Core Service</td>
</tr>
</tbody>
</table>
| Process hacker.exe | 324  | 4    | 0    | 0    | 15.41 MB Microsoft Distributed Transaction Gateway Agent  
| cmd.exe            | 3856 | 1344 | 0.57 | 0    | 5.56 MB VMware Tools Core Service |
| wm.exe             | 528  | 4    | 0    |      | 1.35 MB Local Security Authority Process  
| explorer.exe       | 468  | 468  | 0.57 | 0    | 1.81 MB Windows Logon Application |
| explorer.exe       | 1740 | 9772 | 0.03 | 0    | 24.36 MB Windows Explorer |
| vmtoolsd.exe       | 416  | 416  | 0.07 | 0    | 6.37 KB/s VMware Tools Core Service |
| Process hacker.exe | 324  | 4    | 0    | 0    | 15.41 MB Microsoft Distributed Transaction Gateway Agent  
| cmd.exe            | 3856 | 1344 | 0.57 | 0    | 5.56 MB VMware Tools Core Service |
| wm.exe             | 528  | 4    | 0    |      | 1.35 MB Local Security Authority Process  

**Summary:**
- The `pstree` command is used to list all active processes.
- The table shows various processes with their PIDs, PPIDs, threads, handles, and uptime.
- The processes include system services like `svchost.exe`, `dwm.exe`, and `vmtoolsd.exe`.
- Each process is associated with specific system services and Windows applications.
### psscan
- list hidden and terminated processes; for this image does not produce any output.

### psxview
- cross reference processes with various lists (before using Dementia)

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Name</th>
<th>PID</th>
<th>pslst</th>
<th>psscan</th>
<th>thrdproc</th>
<th>pspcid</th>
<th>csrss</th>
<th>session</th>
<th>deskthrd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7e29ad28</td>
<td>taskeng.exe</td>
<td>3116</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e471030</td>
<td>wininit.exe</td>
<td>404</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7ed39b0</td>
<td>SearchProtocol</td>
<td>2616</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e08490</td>
<td>svchost.exe</td>
<td>876</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e0de518</td>
<td>audiogd.exe</td>
<td>976</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e588748</td>
<td>SearchFilterHo</td>
<td>2636</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e3f77d8</td>
<td>svchost.exe</td>
<td>696</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e5a0c28</td>
<td>GWX.exe</td>
<td>2800</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e104518</td>
<td>svchost.exe</td>
<td>1004</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e4cd7d8</td>
<td>svchost.exe</td>
<td>2232</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e5f59d8</td>
<td>svchost.exe</td>
<td>1460</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e064d28</td>
<td>svchost.exe</td>
<td>824</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e3c56a0</td>
<td>lsass.exe</td>
<td>516</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e19b030</td>
<td>apoolsv.exe</td>
<td>1284</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e7a21a8</td>
<td>SearchIndexer</td>
<td>2540</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e0e9030</td>
<td>svchost.exe</td>
<td>1136</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e3b9d28</td>
<td>services.exe</td>
<td>508</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e34d28</td>
<td>winlogon.exe</td>
<td>468</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e3d1030</td>
<td>ls.exe</td>
<td>524</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e39b308</td>
<td>mdtmpc.exe</td>
<td>2152</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e23ed28</td>
<td>ProcessHacker</td>
<td>2920</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7edeab20</td>
<td>TPAutoConnSvc</td>
<td>240</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e44ad28</td>
<td>FTK Imager.exe</td>
<td>3968</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e100b48</td>
<td>vmtoolsd.exe</td>
<td>1552</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7f0e9158</td>
<td>explorer.exe</td>
<td>1764</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e140338</td>
<td>svchost.exe</td>
<td>1136</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e051778</td>
<td>svchost.exe</td>
<td>784</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e5a5030</td>
<td>appsvc.exe</td>
<td>2832</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e429a0</td>
<td>vmtoolsd.exe</td>
<td>1412</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7fe07910</td>
<td>dllhost.exe</td>
<td>372</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7ef80b36</td>
<td>taskeng.exe</td>
<td>1636</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e50f7e8</td>
<td>VSSVC.exe</td>
<td>2440</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e4c66a8</td>
<td>TPAutoConnect</td>
<td>2224</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e0d4d28</td>
<td>WinPrvSE.exe</td>
<td>252</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e0107a8</td>
<td>svchost.exe</td>
<td>620</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7fc08030</td>
<td>dwm.exe</td>
<td>1716</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7ede1f8</td>
<td>dllhost.exe</td>
<td>636</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7ef7d28</td>
<td>conhost.exe</td>
<td>2244</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e077518</td>
<td>svchost.exe</td>
<td>848</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e47ad28</td>
<td>csrss.exe</td>
<td>412</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e12030</td>
<td>csrss.exe</td>
<td>312</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7f3639e0</td>
<td>smss.exe</td>
<td>228</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7ffa876f0</td>
<td>System</td>
<td>4</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

### psxview
- cross reference processes with various lists (after using Dementia)

<table>
<thead>
<tr>
<th>Offset(P)</th>
<th>Name</th>
<th>PID</th>
<th>pslst</th>
<th>psscan</th>
<th>thrdproc</th>
<th>pspcid</th>
<th>csrss</th>
<th>session</th>
<th>deskthrd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7e009100</td>
<td>svchost.exe</td>
<td>624</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e113600</td>
<td>audiogd.exe</td>
<td>976</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e149b0</td>
<td>DumpIt.exe</td>
<td>3568</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7e4b7f88</td>
<td>SearchFilterHo</td>
<td>2724</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>0x7df08610</td>
<td>dllhost.exe</td>
<td>3312</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
We are not analysing network related activities; therefore, command **netscan** will not produce any viable information.

### 3.6.2.5 Analysis of the results and possible mitigation techniques.

#### 3.6.2.5.1 Interpretation of the results:

- **pstreem** shows that the tool works properly with no visible processes related to ProcessHacker application, the dementia tool, cmd.exe or conhost.exe. There is one thing that makes this image suspicious. Every time we do live acquisition there are remnants of this activity in the memory images – the memory acquisition tool process. In our case, only psxview shows us additional information about that. It means that something suspicious unlinked DumpIt.exe process.

- **psscan** gives us no additional output, and no hidden/unlinked processes were found on the system.

- **psxview** enumerates processes using different techniques. If Dementia attempts to hide processes it will have to evade detection in a number of different ways. Looking at the output, there is one suspicious process called conhost.exe PID: 3576, but further analysis showed that this process was related to the acquisition tool. We can see that Dementia completely evaded capturing our scenario processes. There are no thread processes, no artefacts in PspCidTable, csrss, session structures or desktop threads related to ProcessHacker, Dementia, cmd.exe or conhost.exe. Interesting is that DumpIT.exe process (memory acquisition tool) was unlinked while there are still threads related to it.
At this point only suspicious activity is around the DumpIT.exe process has drawn our attention. Knowing that Dementia needs a special driver in order to hide processes in memory, let us move on to loaded, unloaded and unlinked drivers using modscan.

```bash
vol.py -f w7_dementia_after.raw --profile=Win7SP1x86 modscan
```

<table>
<thead>
<tr>
<th>Offset (P)</th>
<th>Name</th>
<th>Base</th>
<th>Size</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000000007de011c8</td>
<td>secdrv.SYS</td>
<td>0x97549000</td>
<td>0xa000</td>
<td>\SystemRoot\System32\Drivers\secdrv.SYS</td>
</tr>
<tr>
<td>0x000000007de14d68</td>
<td>srvnet.sys</td>
<td>0x97553000</td>
<td>0x2100</td>
<td>\SystemRoot\System32\Drivers\srvnet.sys</td>
</tr>
<tr>
<td>0x000000007de26468</td>
<td>tcpipreg.sys</td>
<td>0x97574000</td>
<td>0xd000</td>
<td>\SystemRoot\System32\Drivers\tcpipreg.sys</td>
</tr>
<tr>
<td>0x000000007df0a9c0</td>
<td>srv.sys</td>
<td>0x96235000</td>
<td>0x52000</td>
<td>\SystemRoot\System32\Drivers\srv.sys</td>
</tr>
<tr>
<td>0x000000007e017130</td>
<td>luafv.sys</td>
<td>0x89895000</td>
<td>0x1b000</td>
<td>\SystemRoot\System32\Drivers\luafv.sys</td>
</tr>
<tr>
<td>0x000000007e078b48</td>
<td>bowser.sys</td>
<td>0x94d8b000</td>
<td>0x19000</td>
<td>\SystemRoot\System32\Drivers\bowser.sys</td>
</tr>
<tr>
<td>0x000000007e2d19c0</td>
<td>DumpIt.sys</td>
<td>0x96316000</td>
<td>0xc0</td>
<td>??\C:\Windows\system32\Drivers\DumpIt.sys</td>
</tr>
<tr>
<td>0x000000007e496f38</td>
<td>cdd.dll</td>
<td>0x94380000</td>
<td>0x1e000</td>
<td>\SystemRoot\System32\cdd.dll</td>
</tr>
<tr>
<td>0x000000007e521768</td>
<td>DementiaKM.sys</td>
<td>0x962fa000</td>
<td>0x1c00</td>
<td>??\C:\DementiaKM.sys</td>
</tr>
<tr>
<td>0x000000007ea3e008</td>
<td>NDProxy.SYS</td>
<td>0x90f33000</td>
<td>0x11000</td>
<td>\SystemRoot\System32\Drivers\NDProxy.SYS</td>
</tr>
<tr>
<td>0x000000007f037e50</td>
<td>usbuhci.sys</td>
<td>0x8ff30000</td>
<td>0xb000</td>
<td>\SystemRoot\system32\Drivers\usbuhci.sys</td>
</tr>
<tr>
<td>0x000000007f046008</td>
<td>USBPORT.SYS</td>
<td>0x8ff3b000</td>
<td>0x4b000</td>
<td>\SystemRoot\system32\Drivers\USBPORT.SYS</td>
</tr>
</tbody>
</table>

We scan services using plugin – svcscan.

```bash
vol.py -f w7_dementia_after.raw --profile=Win7SP1x86 svcscan
```

<table>
<thead>
<tr>
<th>Offset: 0x882a40</th>
<th>Order: 413</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Name: DementiaKM</td>
<td></td>
</tr>
<tr>
<td>Display Name: DementiaKM</td>
<td></td>
</tr>
<tr>
<td>Service Type: SERVICE_KERNEL_DRIVER</td>
<td></td>
</tr>
<tr>
<td>Service State: SERVICE_RUNNING</td>
<td></td>
</tr>
<tr>
<td>Binary Path: \Driver\DementiaKM</td>
<td></td>
</tr>
</tbody>
</table>

We have a suspicious location for a driver – DementiaKM.sys and a service called DementiaKM. The attack can, of course, change this. In order to avoid detection, both the location and the name need to be enhanced. If we try to recover the command history using the cmdscan or consoles plugin we will not get any output, simply because command line processes were hidden by Dementia during memory acquisition. In this case, let’s try to search for artefacts in strings.

First, we have to create Ascii and Unicode strings file.

```bash
srch_strings -a -t d w7_dementia_after.raw > strings.txt
srch_strings -a -t d -el w7_dementia_after.raw >> strings.txt
```

Then, let’s browse strings.txt file for suspicious driver string.

```bash
cat strings.txt | grep DementiaKM.sys
```

249414356 name of the kernel-mode driver (default: 'DementiaKM.sys')
1895632596 name of the kernel-mode driver (default: 'DementiaKM.sys')
134271492 DementiaKM.sys
150102490 DementiaKM.sys
154841104 DementiaKM.sys
164425608 \d\DementiaKM.sys
172129642 DementiaKM.sys
183392104 DementiaKM.sys'
214122210 DementiaKM.sys
216371130 DementiaKM.sys
243016016 DementiaKM.sys
243019150 -d|--driver - name of the kernel-mode driver (default: DementiaKM.sys)
430321208 C:\\DementiaKM.sys
503215912 \Device\HarddiskVolume1\\DementiaKM.sys
cat strings.txt | grep Dementia

162528984 D:\FSEC2012\Dementia-googlecode\Release\SymbolHelper.pdb
249414356 name of the kernel-mode driver (default: 'DementiaKM.sys')
250673608 D:\fsec2012\dementia\dementia-1.0-x64.zip
261026280 d:\fsec2012\dementia\dementia-1.0-x64.zip
503022682 Dementia
503024386 Dementia
508780786 Dementia
742130640 DementiaKMWayDef
1881486792 D:\FSEC2012\Dementia-googlecode\Release\Dementia.pdb
1895632596 name of the kernel-mode driver (default: 'DementiaKM.sys')
2078548546 d:\fsec2012\dementia\dementia-1.0-x64.zip
2107163600 Dementia-1.0-x64.zip
2107163776 Dementia-1.0.zip
2112265272 Dementia.exe
83443200 \Device\HarddiskVolume1\d\Dementia.exe
134271352 Dementia kernel-mode driver
134271492 DementiaKM.sys
134271668 DementiaKM
134271724 Dementia kernel-mode driver
140278736 \Device\HarddiskVolume1\d\Dementia.exe
147957896 \d\Dementia.exe
150102266 Dementia.exe
150102378 DementiaFS.sys
150102490 Dementia.sys
154481104 DementiaKM.sys
154481832 \d\Dementia.exe
164425608 \d\DementiaKM.sys
169514408 The service DementiaKM (DementiaKM) has been created.
169515128 The service DementiaKM (DementiaKM) has been created.
172174744 Dementia.exe
172128613 DementiaFS.sys
172129642 DementiaKM.sys
175263488 DementiaKM
181843364 Administrator: Command Prompt - Dementia.exe -m 2 -a '-p 3324 -p 3856 -P Dementia.exe -P conhost.exe'
183391664 \d\Dementia.exe
183391700 \d\Dementia.exe
183391912 Dementia.exe
183392008 DementiaFS.sys'
183392104 DementiaKM.sys'
189999998 m 2 -a '-p 3324 -p 3856 -P Dementia.exe -P conhost.exe'
194802090 DementiaFS.sys
199059008 Dementia.exe -m 2 -a '-p 3324 -p 3856 -P Dementia.exe -P conhost.exe'
211049176 \d\Device\DementiaKM
214121986 Dementia.exe
214122098 DementiaFS.sys
214122210 DementiaKM.sys
Dementia - v1.0 -- Windows memory anti-forensics suite

Use: Dementia.exe [-h|--help -d|--debug -f|--file] -m <evasion_method> [-a|--args]

DementiaKM

DementiaKM.sys

-Driver - name of the kernel-mode driver (default: DementiaKM.sys)

Dementia-1.0-x64

56 -P Dementia.exe -P conhost.exe'

Administrator: Command Prompt - Dementia.exe -m 2 -a '-p 3324 -p 3856 -P Dementia.exe -P conhost.exe'
The last output (\grep Dementia) gives us additional information about what happened. It also gives us the location of the executable that was used to hide the processes. You can see that those hints are in a lot of different places. Therefore, it is almost impossible to hide it once you have discovered additional hints about a suspicious driver or Windows service. Dementia needs them in order to run properly. Of course name and location can be randomised, but it is still possible to find artefacts. The main problem is that all those artefacts will be connected to Dementia, not to the hidden processes. If this tool succeeds during memory imaging, we will be left with no evidence about what happened to the investigating system.

There is also one additional place where we can find evidence of tampering memory images with this tool. Since we learned from the website that in order to hide processes Dementia uses NtClose and NtWriteFile API functions. We can use apihooks plugin to check if there is something suspicious hooked to those APIs.

Command:

```
vol.py -f w7_dementia_after.raw --profile=Win7SP1x86 apihooks > grep –B 4 –A 25 NtClose
```

will give us additional hints:

```
Hook mode: Kernelmode
Hook type: Inline/Trampoline
Victim module: ntoskrnl.exe (0x82a0a000 - 0x82e23000)
Function: ntoskrnl.exe!NtClose at 0x82c52bf2
Hook address: 0x962fca40
Hooking module: DementiaKM.sys

Disassembly(0):
0x82c52bf2 e9499e6a13    JMP 0x962fca40
0x82c52bf7 51         PUSH ECX
0x82c52bf8 51         PUSH ECX
0x82c52bf9 64a124010000   MOV EAX, [FS:0x124]
0x82c52bff 833d68d4b48200  CMP DWORD [0x82b4d468], 0x0
0x82c52c06 8a         DB 0x8a
0x82c52c07 803a01       CMP BYTE [EDX], 0x1

Disassembly(1):
0x962fca40 8bff        MOV EDI, EDI
0x962fca42 55         PUSH EBP
0x962fca43 8be9        MOV EBX, ESP
0x962fca45 83ec0c       SUB ESP, 0xc
```
We can do the same with NtWriteFile API:

```
vol.py -f w7_dementia_after.raw --profile=Win7SP1x8 apihooks > grep -B 4 -A 25 NtWriteFile
```

*************************************************** *********************
Hook mode: Kernelmode
Hook type: Inline/Trampoline
Victim module: ntoskrnl.exe (0x82a0a000 - 0x82e23000)
Function: ntoskrnl.exe!NtWriteFile at 0x82c7ded2
Hook address: 0x962fc8f0
Hooking module: DementiaKM.sys

Disassembly(0):
0x82c7ded2 e919ea6713    JMP 0x962fc8f0
0x82c7ded7 a6         CMPSB
0x82c7ded8 82e87a       SUB AL, 0x7a
0x82c7dedb bee0ff33f6    MOV ESI, 0xf633ffe0
0x82c7dee0 8975dc       MOV [EBP-0x24], ESI
0x82c7dee3 8975d0       MOV [EBP-0x30], ESI
0x82c7dee6 8975a4       MOV [EBP-0x5c], ESI
0x82c7dee9 89         DB 0x89

Disassembly(1):
0x962fc8f0 8bff        MOV EDI, EDI
0x962fc8f2 55         PUSH EBP
0x962fc8f3 8bec        MOV EBP, ESP
0x962fc8f5 83ec0c       SUB ESP, 0xc
0x962fc8f8 837d0c00     CMP DWORD [EBP-0xc], 0x0
0x962fc8fc 0f8505010000   JNZ 0x962fca07
0x962fc902 837d1000     CMP DWORD [EBP-0x10], 0x0
0x962fc906 0f DB 0xf
0x962fc907 85 DB 0x85

Key remarks:

- First of all we should start with identification of our imaging tool among running processes, if we cannot find it using the pstree plugin then that is a strong indicator that something unlinked this process in order to modify the output.
- The best option is to search for a solution is to search for Dementia indicators using the apihooks plugin. If there is something hooked to NtClose and NtFileWrite API functions we should definitely investigate it.
- We should always check for possible artefacts in the strings, but, first of all, we need at least hints about suspicious services or drivers. We will not always get results from strings as they can be randomly overwritten.
- Dementia has the ability to hide itself. In order to achieve that it has to be run with a ‘–P [Dementia executable name]’ argument.
- Another possible approach to detect Dementia would be to capture all the processes using Process Hacker or Process Explorer during live acquisition and then compare it with Volatility pslist, psxview plugins against captured memory image. If Dementia is in the system, we may observe some missing processes.
- The tool relies on various heuristics of acquisition tools like process name, driver, file object values and flags (see table below). In order to be completely resilient, we would have to randomise all these ‘patterns’ and create custom memory imaging tool.
To summarise, the Dementia tool currently renders most memory acquisition applications completely vulnerable, although it is quite easy to detect. Keeping in mind that this project is open-source, the final version will be customised by possible attackers and possibly will become more difficult to find. Once it detects the acquisition process, the captured memory image will be stripped of any relevant evidence. Additionally, it requires full administrative rights in order to inject additional drivers, just like the ADD tool. Analysing the build signature of the executable or Dementia drivers in virustotal.com gives no indicators whether or not these files are suspicious or harmful; therefore, they will not be picked up by any antivirus software.

### 3.7 Microsoft anti-forensic settings

#### 3.7.1 Leave no trace in Windows

The Windows Operating System (OS) contains a multitude of different types of artefacts that are very useful during the forensic investigation and reveal suspicious activities. This chapter contains general counter-forensic settings that minimise digital footprints and prevent an investigator from proving what has been done even if the forensic examiner has, for example, access to the compromised computer.[33]

#### 3.7.2 Disabling timestamps

Using the timestamps, a forensic analyst can build a timeline that can be very helpful when cross-referenced with other known device. Disabling this functionality makes the investigation harder. There are a variety of ways of doing it.

1. Disable **UserAssist** logging.
   - **UserAssist** is a registry key that keeps dates/times, count and path to executable of the applications launched by a user.
   - Create a new key named ‘Settings’ in **UserAssist**. In this key create DWORD value named **NoLog**, set the value to 1.

2. Disable prefetching.
   - **Disabled service** **Superfetch** (Service name is **SysMain**).
3. Disable last access time.
   • fsutil behavior set DisableLastAccess 1 (host has to be restarted).
   • Configure HKLM\SYSTEM\CurrentControlSet\Control\FileSystem key and NtfsDisableLastAccessUpdate
   • Default settings for Win7/Win8 is 1 (disabled).

3.7.3 Additional settings

3.7.3.1 Disable hibernation

This functionality can be configured through the Control Panel graphical user interface (Power options) or using the command `powercfg /hibernate on` (or off). Another way to do it is to change registry values in HKLM\SYSTEM\CurrentControlSet\Control\Power for both HiberFileSizePercent and HibernateEnabled.

3.7.3.2 Delete the history of USB activities

USBSTOR registry key contains the history of USB devices. To hide the use of a USB device, the content of the HKLM\SYSTEM\CurrentControlSet\Enum\USBSTOR key is often deleted. Another artefact that stores information about USB activity is setupapi.dev.log file, which is located in C:\Windows\INF folder.

3.7.3.3 Disable System Restore point (or Volume Shadow Copy)

A change to the Restore settings can be done through System Protection tab in System Properties GUI. It is possible to delete a specific Volume Shadow Copy using the command `vssadmin delete shadows`.

To avoid the presence of suspicious files in Volume Shadow Copy (VSS), it is possible to exclude files from Shadow Copies. A VSS application can delete files from a shadow copy during shadow copy creation by using the following registry key HKLM\SYSTEM\CurrentControlSet\Control\BackupRestore\FilesNotToSnapshot. The files are specified by fully qualified paths.[34]

3.7.3.4 Disable debugging upon failure or application crash

To disable investigator to get information from pop-up dialog windows after application failure or crash, the following has to be configured:

   • Set registry to 1 for value DontShowUI in HCU\Software\Microsoft\Windows\Windows Error Reporting.

3.7.3.5 Disable Windows Event logging

Windows OS generates many event logs and they very often contain crucial information for an investigation. Disabling logging or deleting event logs is one of the basic techniques for frustrating incident responder or investigator. To disable Windows Event Log service:

   • From command line:
     ```
     REG add ‘HKLM\SYSTEM\CurrentControlSet\services\eventlog’ /v Start /t REG_DWORD /d 4 /f
     ```
   • Or using msconfig, go to Services tab and locate Windows Event Log. Then uncheck the box to prevent this service to start.
• Or disable it directly in `services.msc` console find `Windows Event Log service` and in service properties change the startup type to `Disabled`.

1. Delete content of the `pagefile.sys` file before the system is turned off or delete file [35]
   • Change the data value of the `ClearPageFileAtShutdown` value in the following registry key to a value of 1: `HKLM\System\CurrentControlSet\Control\Session Manager\Memory Management`.
   • Or disable it from Advanced System Settings at the Advanced tab.

### 3.7.4 Key remarks

To configure most of these settings requires administrative privileges, and to apply them a restart is usually needed. Detection of suspicious settings can be achieved, for example, by checking or listing values from the registry using pre-prepared script during the incident response process.

`CCleaner` is actually an example of an anti-forensic tool for Windows. This application can be used for wiping important evidence of any malicious activity. Therefore, the discovery of any artefact related to CCleaner may also be a proof of possible anti-forensic activity.
4 Conclusions

Recently, anti-forensic techniques have been the subject of much discussion. Some has tried to make us believe that the forensic process is in the state of decay. Others to convince us to remain in the blessed state of confidence that the forensic tools and expert’ knowledge will solve everything no matter how sophisticated were the techniques thrown against the investigating system. In our paper, we have tried to stay somewhere in the middle. Starting from timestamp manipulation tools, through data hiding, DOS attacks on DFTs and use of external media and RAM memory based tools, we have shown how powerful they can be in the hands of skilled criminals. These are just the most known examples which can be found on the internet. Our goal was also to explain how host-related anti-forensic tools work in practice, and the major challenges to fight them. We saw one area of common ground to remain one step ahead of them – live acquisition and memory analysis. There is no need to explain how beneficial it may become to use strings, especially against memory images or volatility plugins like cmdscan or consoles. We have to be aware that some malicious activities will only become visible in the memory since they do not touch the hard drive at all. But there are also drawbacks. We cannot completely rely on memory analysis without double checking the output and the final result that is the memory image. We presented tools which might tamper with the evidence before we even started analysing it. Therefore, there is a strong need for custom imaging tools and specific procedures to avoid the situation where we are left with evidence that is no use to the case we are working on. There are some hints in this paper how to prepare for such moments.

For the future work, since all stolen data has to be exfiltrated, either physically or over a network, we should also investigate techniques like steganography or network related tools that allow the attackers to remain undetected and transfer acquired information to a place where it can be used. This is the field where we can be even more successful with the right use of forensic tools. If we are able to break the last phase of the cyber kill chain then the attack was not successful and the integrity of our systems can be easily restored.

Finally, in our paper, all the anti-forensic techniques were analysed one by one. For a better understanding of how they can be implemented in the real world, it would be advisable to create a separate scenario that combines most of them into one multi-layered case. This will show how much complexity the combination brings.
References


[27] hephaest0s, ‘usbkill,’ [Online].
https://github.com/hephaest0s/usbkill


http://blog.handlerdiaries.com/?p=363


https://code.google.com/p/dementia-forensics/

http://haxf4rall.com/tag/anti-forensics/

[34] Microsoft Developer Network. [Online].

https://support.microsoft.com/en-us/kb/314834