CCDCOE

NATO Cooperative Cyber Defence Centre of Excellence Tallinn, Estonia

Patrycjusz Zdzichowski, Michal Sadlon, Teemu Uolevi Väisänen

Alvaro Botas Munoz, Karina Filipczak

Anti-Forensic Study

Tallinn 2015

www.ccdcoe.org
publications@ccdcoe.org

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

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

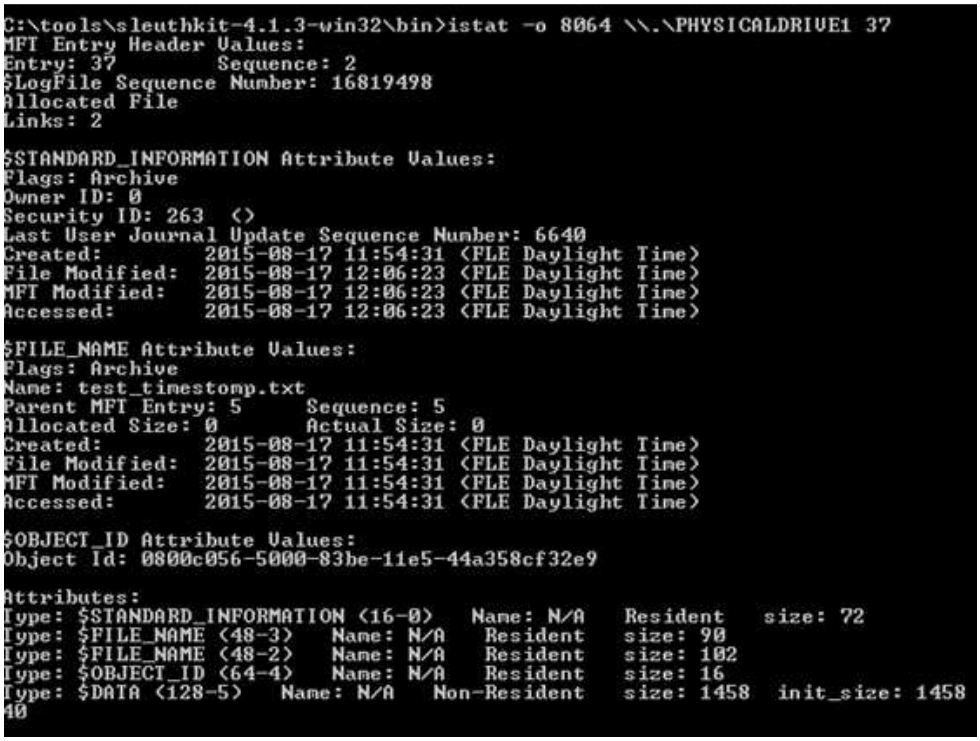
Time/date format: ‘DayOfWeek MMDDYYYY HH:MM:SS [AM|PM]’

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’

```
C:\tools\sleuthkit-4.1.3-win32\bin>istat -o 8064 \\.\PHYSICALDRIVE1 37
MFT Entry Header Values:
Entry: 37        Sequence: 2
$LogFile Sequence Number: 16820389
Allocated File
Links: 2

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 263  ()
Last User Journal Update Sequence Number: 6832
Created:        2010-10-10 10:10:00 (FLE Daylight Time)
File Modified:  2010-10-10 10:10:00 (FLE Daylight Time)
MFT Modified:   2010-10-10 10:10:00 (FLE Daylight Time)
Accessed:       2010-10-10 10:10:00 (FLE Daylight Time)

$FILE_NAME Attribute Values:
Flags: Archive
Name: test_timestomp.txt
Parent MFT Entry: 5      Sequence: 5
Allocated Size: 0        Actual Size: 0
Created:        2015-08-17 11:54:31 (FLE Daylight Time)
File Modified:  2015-08-17 11:54:31 (FLE Daylight Time)
MFT Modified:   2015-08-17 11:54:31 (FLE Daylight Time)
Accessed:       2015-08-17 11:54:31 (FLE Daylight Time)

$OBJECT_ID Attribute Values:
Object Id: 0800c056-5000-83be-11e5-44a358cf32e9

Attributes:
Type: $STANDARD_INFORMATION (16-0)   Name: N/A   Resident   size: 72
Type: $FILE_NAME (48-3)   Name: N/A   Resident   size: 90
Type: $FILE_NAME (48-2)   Name: N/A   Resident   size: 102
Type: $OBJECT_ID (64-4)   Name: N/A   Resident   size: 16
Type: $DATA (128-5)   Name: N/A   Non-Resident   size: 1458  init_size: 1458
40
```

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:

FileName	USN	Timestamp	Reason	MFTReference	FileAttributes
test_timestomp.txt	6736	2015-08-17 13:19:54.397:2414	BASIC_INFO_CHANGE	37	archive
test_timestomp.txt	6832	2015-08-17 13:19:54.397:2414	BASIC_INFO_CHANGE+CLOSE	37	archive

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.

UpdateNonResidentValue	Noop	test_timestomp.txt	$DATA:$J	;$UsnJrnl	37	2015-08-17 13:19:54.397:2414	BASIC_INFO_CHANGE	6736
UpdateNonResidentValue	Noop	test_timestomp.txt	$DATA:$J	;$UsnJrnl	37	2015-08-17 13:19:54.397:2414	BASIC_INFO_CHANGE+CLOSE	6832

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.

lf_RedoOperation	lf_UndoOperation	lf_FileName	lf_CurrentAttribute	lf_SI_CTime	lf_SI_ATime	lf_SI_MTime	lf_SI_RTime
UpdateResidentValue	UpdateResidentValue	test_timestomp.txt	$STANDARD_INFORMATION	2010-10-10 10:10:00.000:0000	2010-10-10 10:10:00.000:0000	2010-10-10 10:10:00.000:0000	2010-10-10 10:10:00.000:0000

It is possible to detect the timestomp 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:MSMSMS:NSNSNSNS. 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\sleuthkit-4.1.3-win32\bin>istat -o 8064 \\.\PHYSICALDRIVE1 38
MFT Entry Header Values:
Entry: 38        Sequence: 2
$LogFile 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
Created:        2015-08-17 11:54:49 (FLE Daylight Time)
File Modified:  2015-08-17 12:05:56 (FLE Daylight Time)
MFT Modified:   2015-08-17 12:05:56 (FLE Daylight Time)
Accessed:       2015-08-17 12:05:56 (FLE Daylight Time)

$FILE_NAME Attribute Values:
Flags: Archive
Name: test_setmace.txt
Parent MFT Entry: 5     Sequence: 5
Allocated Size: 0       Actual Size: 0
Created:        2015-08-17 11:54:49 (FLE Daylight Time)
File Modified:  2015-08-17 11:54:49 (FLE Daylight Time)
MFT Modified:   2015-08-17 11:54:49 (FLE Daylight Time)
Accessed:       2015-08-17 11:54:49 (FLE Daylight Time)

$OBJECT_ID Attribute Values:
Object Id: 0800c056-5000-83be-11e5-44a358cf32e7

Attributes:
Type: $STANDARD_INFORMATION (16-0)   Name: N/A   Resident   size: 72
Type: $FILE_NAME (48-3)   Name: N/A   Resident   size: 90
Type: $FILE_NAME (48-2)   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
39
```

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

```
C:\tools\sleuthkit-4.1.3-win32\bin>istat -o 8064 \\.\PHYSICALDRIVE1 38
MFT Entry Header Values:
Entry: 38        Sequence: 2
$LogFile 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
Created:        2010-10-10 13:10:00 (FLE Daylight Time)
File Modified:  2010-10-10 13:10:00 (FLE Daylight Time)
MFT Modified:   2010-10-10 13:10:00 (FLE Daylight Time)
Accessed:       2010-10-10 13:10:00 (FLE Daylight Time)

$FILE_NAME Attribute Values:
Flags: Archive
Name: test_setmace.txt
Parent MFT Entry: 5     Sequence: 5
Allocated Size: 0       Actual Size: 0
Created:        2010-10-10 13:10:00 (FLE Daylight Time)
File Modified:  2010-10-10 13:10:00 (FLE Daylight Time)
MFT Modified:   2010-10-10 13:10:00 (FLE Daylight Time)
Accessed:       2010-10-10 13:10:00 (FLE Daylight Time)

$OBJECT_ID Attribute Values:
Object Id: 0800c056-5000-83be-11e5-44a358cf32e7

Attributes:
Type: $STANDARD_INFORMATION (16-0)   Name: N/A   Resident   size: 72
Type: $FILE_NAME (48-3)   Name: N/A   Resident   size: 90
Type: $FILE_NAME (48-2)   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
39
```

- 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.

FileName	USN	Timestamp	Reason	MFTReference	FileAttributes
sectorio64.sys	2061164992	2015-08-14 13:17:04.625:2289	FILE_CREATE	65716	archive
sectorio64.sys	2061165080	2015-08-14 13:17:04.625:2289	DATA_EXTEND+FILE_CREATE	65716	archive
sectorio64.sys	2061165168	2015-08-14 13:17:04.635:2290	CLOSE+DATA_EXTEND+FILE_CREATE	65716	archive
sectorio64.sys	2061165872	2015-08-14 13:17:06.037:7310	CLOSE+FILE_DELETE	65716	archive
sectorio64.sys	2061166200	2015-08-14 13:17:07.210:2326	FILE_CREATE	65716	archive
sectorio64.sys	2061166288	2015-08-14 13:17:07.210:2326	DATA_EXTEND+FILE_CREATE	65716	archive
sectorio64.sys	2061166376	2015-08-14 13:17:07.230:2327	CLOSE+DATA_EXTEND+FILE_CREATE	65716	archive
sectorio64.sys	2061166464	2015-08-14 13:17:07.280:2327	CLOSE+FILE_DELETE	65716	archive

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.

```
**************************************************
CommandProcess: conhost.exe Pid: 4884
CommandHistory: 0x38edd0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 3 LastAdded: 2 LastDisplayed: 2
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x64
Cmd #0 @ 0x38dae0: cd \
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 \
  C:\>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:

```
***************************************************************************
MFT entry found at offset 0x522c6400
Attribute: In Use & File
Record Number: 16925
Link count: 1


$STANDARD_INFORMATION
Creation                    Modified                    MFT Altered                 Access Date                 Type
--------------------------- --------------------------- --------------------------- --------------------------- ----
2015-09-07 13:31:21 UTC+0000 2015-08-06 11:56:20 UTC+0000   2015-09-07 13:31:22 UTC+0000   2015-09-08 08:44:21 UTC+0000   Archive

$FILE_NAME
Creation                    Modified                    MFT Altered                 Access Date                 Name/Path
--------------------------- --------------------------- --------------------------- --------------------------- ---------
2015-09-07 13:31:21 UTC+0000 2015-09-07 13:31:21 UTC+0000   2015-09-07 13:31:21 UTC+0000   2015-09-07 13:31:21 UTC+0000   Af\SetMace\SetMace.exe
```

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:

```
Last Modified         Path
------------------------------ ----
….
….
2015-08-06 11:56:20 UTC+0000 \??\C:\Af\SetMace\SetMace.exe
2015-07-29 17:57:10 UTC+0000
??\C:\Windows\SoftwareDistribution\Download\Install\mpsyschk.exe
….
```

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.

- *Timeline analysis* - might also reveal anomalies within the system.
- *MFT entry and sequence number* – tells if the file is outside the range that it should be.

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 timestomp 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

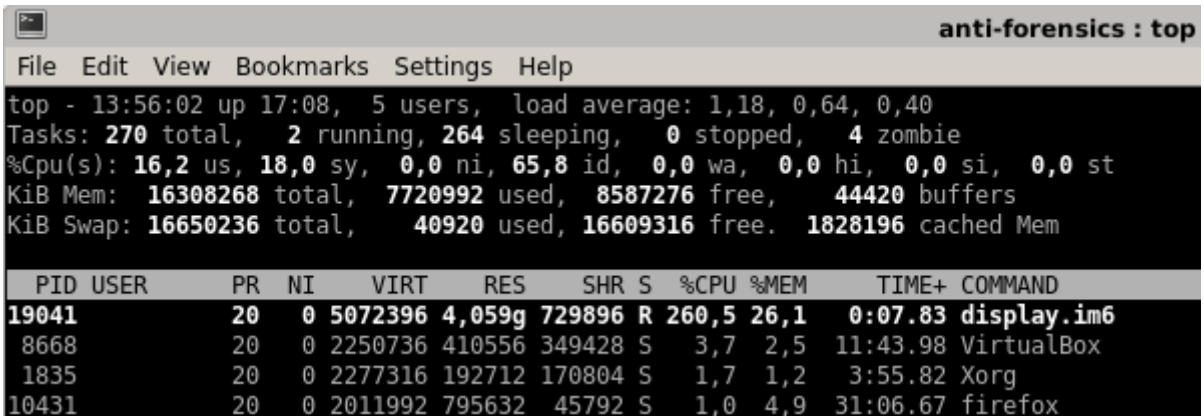
3.2.3.1 Crafted images

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].

```
$ sha256sum picture-1G-19000x19000.png
50d5587d5b1d62382450959ee202862808e7a944603ccf3b8c9006fb1e02d1a9 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
File Name                     : picture-1G-19000x19000.png
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
0000010 0000 384a 0000 384a 0301 0000 4600 c2d4
0000020 0001 0000 5003 544c ff45 0000 e219 3709
...
```

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: db6981082063dbb4bac89d27c41fbeb86d9e4a97b36661c0945b77a6b9bb0948), 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.

Files	**Size after full uncompression**	**Notes**
42.zip	4,5 PB	42.zip includes 16 files: lib0.zip - libf.zip, each 34,902 bytes (34,9 kB)
+-- lib 0.zip ... lib f.zip	4,5 TB	Each includes 16 files: book 0.zip - book f.zip, each 29,446 bytes (29,4 kB)
+-- book 0.zip ... book f.zip	281,5 TB	Each includes 16 files: chapter 0.zip - chapter f.zip, each 32,150 bytes (32,1 kB)
+-- chapter 0.zip ... chapter f.zip	17,6 TB	Each includes 16 files: doc 0.zip - doc f.zip, each 164,302 bytes (165,3 kB)
+-- doc 0.zip ... doc f.zip	1,1 TB	Each includes 16 files: page 0.zip - page f.zip, each 4,168,266 bytes (4,2 MB)
+-- page 0.zip ... page f.zip	68,7 GB	Each includes 1 file: 0.dll which size is 4,294.967,295 bytes (4,3 GB)
+ -- 0.dll	4,294.967,295 bytes (4,3 GB)	Includes just 0xAA

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
```

lib 3.zip (QDz 373y J%y? 4Jj% NO[{0t XsL~ kW=5 MQndO x7k0~5 M}<. +Ogu _wFy Z&-4_ @t90	8[q}v l/vk xr>A >/}X Wnyc gx wz3N Ou2! f(3’ q5iD C,SD #pP* Rjr[%]1i O'N% DTB1E GLI~e =W7._7B $:(A S3;dQ b?_d zuI:&y <_}L {aP'I :!3Hw	LXB , K0a 5 ` &, lib 1.zip (QDz 373y J%y? 4Jj% NO[{0t XsL~. kW=5 MQndO x7k0~5 M}<. +Ogu _wFy Z&-4_ @t90 8[q}v l/vk xr>A >/}X Wnyc	gx wz3N Ou2! f(3’ q5iD C,SD #pP* Rjr[%]1i O'N% DTB1E GLI~e =W7._7B $:(A S3;dQ b?_d zuI:&y <_}L {aP'I :!3Hw LXB , K0a 5 ` &, <clip>

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 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 - Structure of Harmless_4_ZettaByte_ZIP-bomb.zip.

Files	Size after full uncompression	Notes
literature	?	literature includes 1 libraries
+-- libraries	?	Each includes 32 libraries
+-- library	4,7 ZB	Each includes 32 volumes
+-- volume	147,6,4 EB	Each includes 32 books
+-- book	4,6 EB	Each includes 32 chapters
+-- chapter	144,1 PB	Each includes 32 parts
+-- part	4,5 PB	Each includes 32 sections
+-- section	140,7 TB	Each includes 32 pages
+-- page	4,4 TB	Each includes 32 columns
+-- column	137,4 GB	Each includes 1 .dll file
+-- .dll	4,29 GB	Includes zeros

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.

```
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, 1CPU, 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

Slacker application downloaded from http://www.jbbrowning.com/sandbox/security.html.

Tool syntax:

```
Hiding a file in slack space:
------------------------------
slacker.exe -s <file> <path> <levels> <metadata> [password] [-dxi] [-n|-k|-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
-nkf                    none, random key, or file based data obfusaction
<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
```

```
C:\AF\slacker>slacker.exe -s c:\secret.txt m:\Files 1 c:\AF\slacker\test.jpg -d -n
Mode: store
File: c:\secret.txt
Path: m:\Files
Levels: 1
Meta: c:\AF\slacker\test.jpg
Pass: (null)
Tech: 1
Hide: 1
File being hidden: c:\secret.txt
Filename length: 14
File size: 44
Xor Key: 0
Number of victim files used: 1


File index: 0
Filename: m:\Files\Chrysanthemum.jpg
Filename length: 27
Last known file size: 879394
Used sectors: 1
```

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 *Chrysanthenum.jpg* file was used in this case to store data from *secret.txt* file.

3.3.1.2 Analysis and possible mitigation techniques

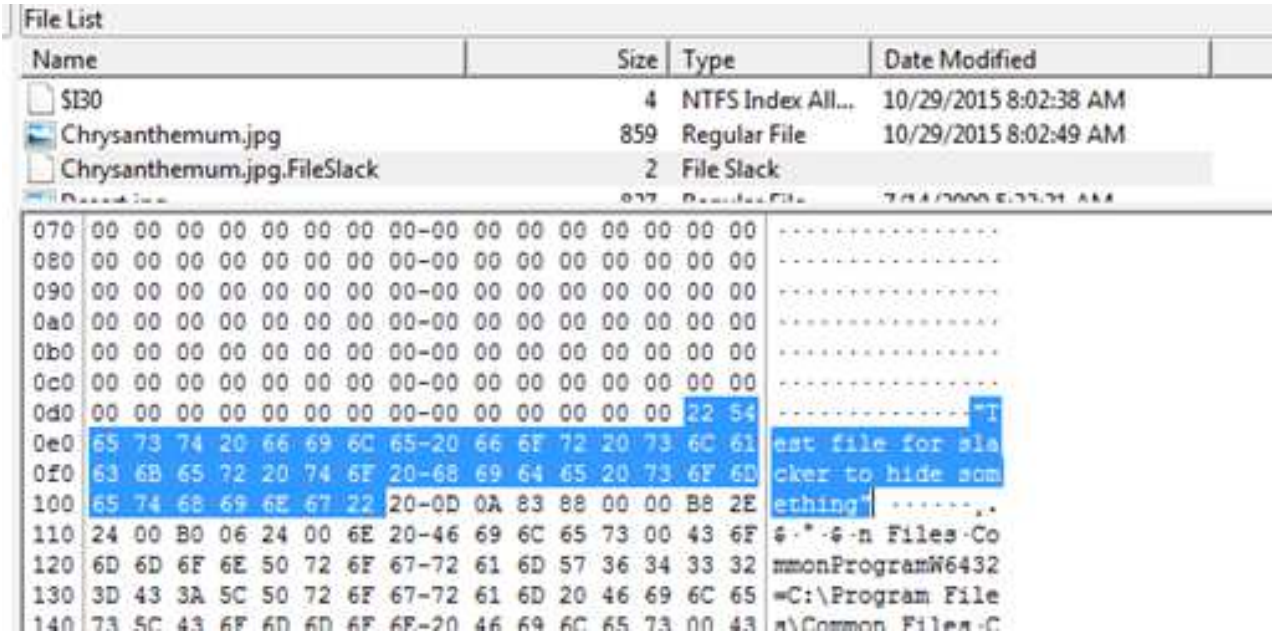
NTFS $LogFile and change journal ($UsnJrnl) files were exported after the slacker application testing. Output from the NTFS change journal related to slacker application activity:

FileName	USN	Timestamp	Reason	MFTReference
Chrysanthemum.jpg	2668920	2015-10-29 11:02:49.182:5210	DATA_EXTEND	2024
Chrysanthemum.jpg	2669016	2015-10-29 11:02:49.182:5210	DATA_EXTEND+DATA_TRUNCATION	2024
Chrysanthemum.jpg	2669112	2015-10-29 11:02:49.198:1210	CLOSE+DATA_EXTEND+DATA_TRUNCATION	2024

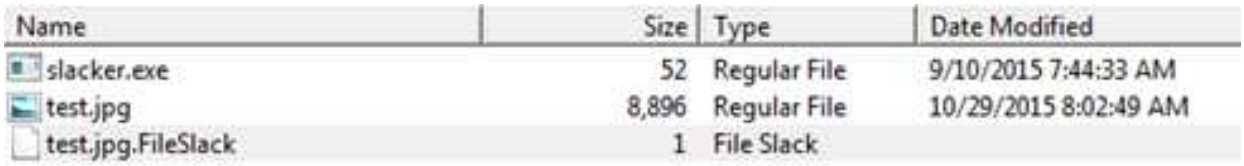
Reason explanation:

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

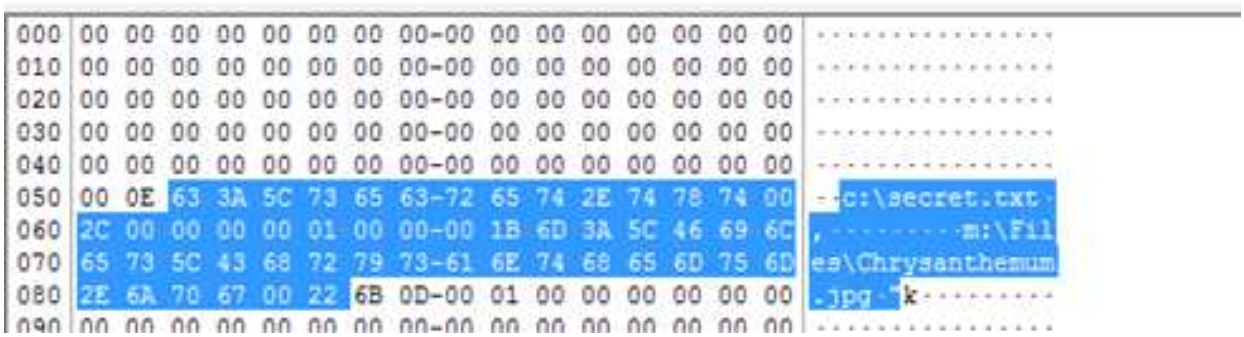
Most forensic tools are able to read data from file slack space and in this case, Access Data FTK Imager was used to check data stored in output file slack.



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.



Name	Size	Type	Date Modified
slacker.exe	52	Regular File	9/10/2015 7:44:33 AM
test.jpg	8,896	Regular File	10/29/2015 8:02:49 AM
test.jpg.FileSlack	1	File Slack	



```
000 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
050 00 0E 63 3A 5C 73 65 63-72 65 74 2E 74 78 74 00 ..c:\secret.txt.
060 2C 00 00 00 00 01 00 00-00 1B 6D 3A 5C 46 69 6C ,.........m:\Fil
070 65 73 5C 43 68 72 79 73-61 6E 74 68 65 6D 75 6D es\Chrysanthemum
080 2E 6A 70 67 00 22 6B 0D-00 01 00 00 00 00 00 00 .jpg."k.........
090 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

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
```

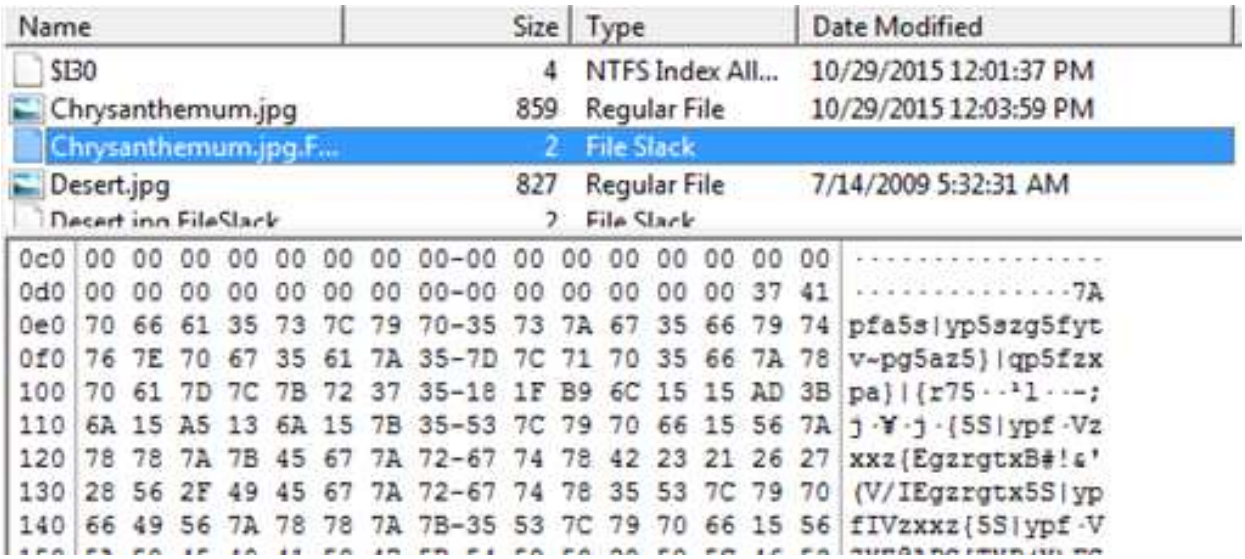
```
C:\AF\slacker>slacker.exe -s c:\secret.txt m:\Files 1 c:\af\slacker\test2.jpg password -x -k
Mode: store
File: c:\secret.txt
Path: m:\Files
Levels: 1
Meta: c:\af\slacker\test2.jpg
Pass: password
Tech: 2
Hide: 2
File being hidden: c:\secret.txt
Filename length: 14
File size: 44
Xor Key: 15
Number of victim files used: 1

File index: 0
Filename: m:\Files\Chrysanthemum.jpg
Filename length: 27
Last known file size: 879394
Used sectors: 1
```

Where:

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

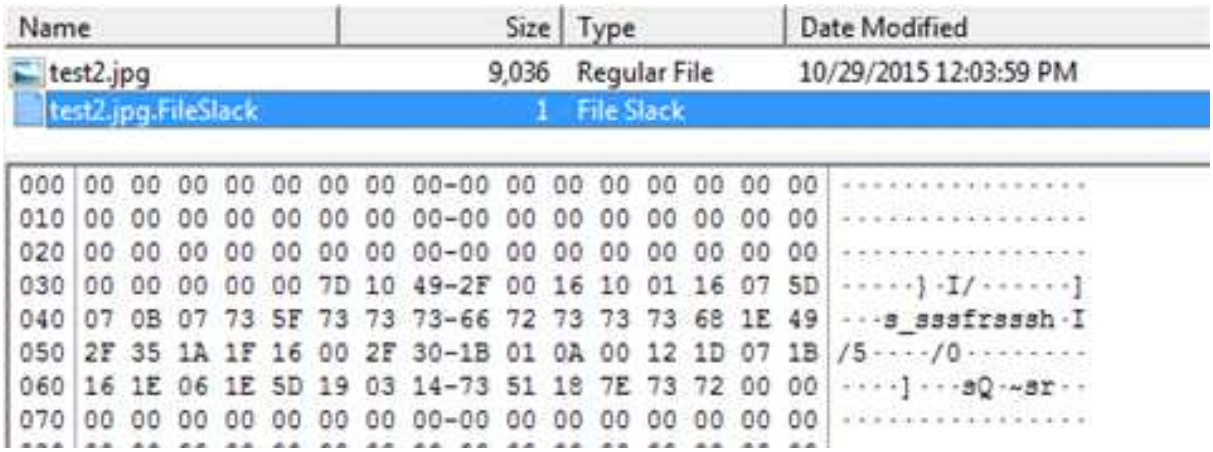
Slack space of the output file does not contain clear text information as visible below.



Name	Size	Type	Date Modified
$I30	4	NTFS Index All...	10/29/2015 12:01:37 PM
Chrysanthemum.jpg	859	Regular File	10/29/2015 12:03:59 PM
Chrysanthemum.jpg.F...	2	File Slack	
Desert.jpg	827	Regular File	7/14/2009 5:32:31 AM
Desert.jpg FileSlack	2	File Slack	

```
0c0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ----------------
0d0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 37 41  --------------7A
0e0  70 66 61 35 73 7C 79 70-35 73 7A 67 35 66 79 74  pfa5s|yp5szg5fyt
0f0  76 7E 70 67 35 61 7A 35-7D 7C 71 70 35 66 7A 78  v~pg5az5}|qp5fzx
100  70 61 7D 7C 7B 72 37 35-18 1F B9 6C 15 15 AD 3B  pa}|{r75··¹l··-;
110  6A 15 A5 13 6A 15 7B 35-53 7C 79 70 66 15 56 7A  j·¥·j·{5S|ypf·Vz
120  78 78 7A 7B 45 67 7A 72-67 74 78 42 23 21 26 27  xxz{EgzrgtxB#!&'
130  28 56 2F 49 45 67 7A 72-67 74 78 35 53 7C 79 70  (V/IEgzrgtx5S|yp
140  66 49 56 7A 78 78 7A 7B-35 53 7C 79 70 66 15 56  fIVzxxz{5S|ypf·V
```

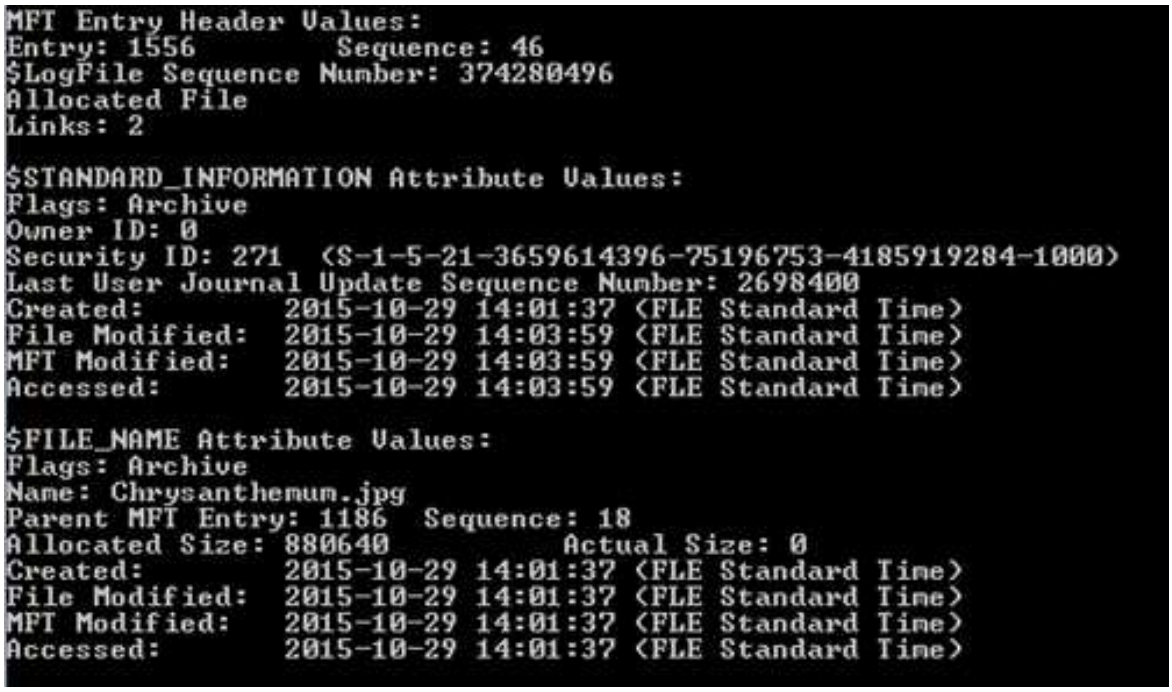
And also password was used to obfuscate metadata therefore also this information is not comprehensible for investigator.



Name	Size	Type	Date Modified
test2.jpg	9,036	Regular File	10/29/2015 12:03:59 PM
test2.jpg.FileSlack	1	File Slack	

```
000  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ----------------
010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ----------------
020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ----------------
030  00 00 00 00 00 7D 10 49-2F 00 16 10 01 16 07 5D  -----}·I/------]
040  07 0B 07 73 5F 73 73 73-66 72 73 73 73 68 1E 49  ···s_sssfrssh·I
050  2F 35 1A 1F 16 00 2F 30-1B 01 0A 00 12 1D 07 1B  /5----/0--------
060  16 1E 06 1E 5D 19 03 14-73 51 18 7E 73 72 00 00  ----]···sQ·~sr··
070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ----------------
```

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):

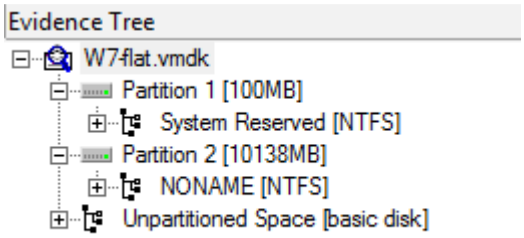
MD5 sum - W7-flat.vmdk: 1714f0435fbb069e7c20c7ff518212fc

Hard drive after the test (Puppy Linux 7.0.3):

MD5 sum - W7-flat.vmdk: 763ba0f2884e3d9293ffce85589d7c8b

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.

W7-flat.vmdk reference image		W7-flat.vmdk – after test – Ubuntu	
File name	MD5 sum	File name	MD5 sum
\[root]\$I30	efc01c2615639a7523029e212f6716a5	\[root]\$I30	8bff22bb225d6e3d785d90a1518927e1
[root]\Users\$I30	ef0482bd03243d53fc649022f4fefe17	[root]\Users\$I30	0c063ff64a63292e5e783b9ee4032f74
[root]\Users\gib5on\$I30	7c97fcce12a939ece40eb6b77d1afd21	[root]\Users\gib5on\$I30	167c8b9744547c86db3193ecdb30a072
[root]\Users\gib5on\AppData\Roaming\Microsoft\Windows\$I30	e96256a782e8914338141d578a3da8bf6	[root]\Users\gib5on\AppData\Roaming\Microsoft\Windows\$I30	84bfe0aec6bcdb1c2660073d1d786e73
File not present!		[root]\Users\gib5on\Desktop\$I30	d95714f405a4bfc05c359fa82fdac45e
[root]\Users\gib5on\Desktop\secret.txt	b85eef11e0f37522bf5ce4ce45b790ca	[root]\Users\gib5on\Desktop\secret.txt	ab1c3a815c3398f0a40ecac6b27c6006
[root]\$Bitmap	1060900783691d2921eb395a0f90d7a1	[root]\$Bitmap	a0af1b7ff05888c2a19da526d00eec3d
[root]\$MFT	dfd54a71ca5376e6835ad783e2da94f6	[root]\$MFT	7e721739d5b0daae574f4e1e424d957b
[root]\Windows\ServiceProfiles\LocalService\AppData\Local\lastalive1.dat	69ed9bdca7be163777c071e571e89fb8	[root]\Windows\ServiceProfiles\LocalService\AppData\Local\lastalive1.dat	c99a74c555371a433d121f551d6c6398
[unallocated space]\1183483	75a1e3aee1ad87785d2dee39c3883e8b	File not present!	

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

W7-flat.vmdk reference image		W7-flat.vmdk – after test – Ubuntu	
File name	ACCESSED TIME	File name	ACCESSED TIME
autoexec.bat	2015-08-31 07:12:43.621071	autoexec.bat	**2015-09-01 13:37:48.057432**
config.sys	2015-08-31 07:12:43.620800	config.sys	**2015-09-01 13:37:48.057743**
Program Files	2011-04-12 01:34:51.409618	Program Files	**2015-09-01 13:37:48.096691**
ProgramData	2009-07-14 04:53:55.611240	ProgramData,	**2015-09-01 13:37:48.092342**
PROGRA~1	2011-04-12 01:34:51.409618	PROGRA~1	**2015-09-01 13:37:48.096691**
PROGRA~2	2009-07-14 04:53:55.611240	PROGRA~2	**2015-09-01 13:37:48.092342**

Second location - \[root]\Users\$I30

W7-flat.vmdk reference image		W7-flat.vmdk – after test – Ubuntu	
File name	ACCESSED TIME	File name	ACCESSED TIME
Default	2009-07-14 07:17:20.049328	Default	**2015-09-01 13:37:49.447098**
desktop.ini	2009-07-14 04:41:57.452887	desktop.ini	**2015-09-01 13:37:49.429901**
Public	2011-04-12 01:34:44.365992	Public	**2015-09-01 13:37:49.455217**

Third location - \[root]\Users\gib5on\$I30

W7-flat.vmdk reference image		W7-flat.vmdk – after test – Ubuntu	
File name	ACCESSED TIME	File name	ACCESSED TIME
Contacts	2015-08-31 11:56:03.858677	Contacts	**2015-09-01 13:37:51.139742**
Desktop	2015-08-31 13:18:40.689661	Desktop	**2015-09-01 13:39:05.077579**
Downloads	2015-08-31 11:56:09.006687	Downloads	**2015-09-01 13:37:51.188358**
DOWNLO~1	2015-08-31 11:56:09.006687	DOWNLO~1	**2015-09-01 13:37:51.188358**
Music	2015-08-31 11:56:08.975487	Music	**2015-09-01 13:37:51.194149**
ntuser.dat.LOG1	2015-08-31 11:56:01.347073	ntuser.dat.LOG1	**2015-09-01 13:37:51.124277**
ntuser.ini	2015-08-31 11:56:01.378273	ntuser.ini	**2015-09-01 13:37:51.135002**
Pictures	2015-08-31 11:56:08.975487	Pictures	**2015-09-01 13:37:51.195230**
Videos	2015-08-31 11:56:08.975487	Videos	**2015-09-01 13:37:51.196953**

W7-flat.vmdk reference image		W7-flat.vmdk – after test – Ubuntu	
File name	ACCESSED TIME	File name	ACCESSED TIME
Cookies	2015-08-31 11:56:09.849089	Contacts	2015-09-01 13:37:51.173048
Network Shortcuts	2015-08-31 11:56:01.331472	Desktop	2015-09-01 13:37:51.171473
NETWOR~1	2015-08-31 11:56:01.331472	Downloads	2015-09-01 13:37:51.171473
Printer Shortcuts	2015-08-31 11:56:01.315872	DOWNLO~1	2015-09-01 13:37:51.170448
PRINTE~1	2015-08-31 11:56:01.315872	Music	2015-09-01 13:37:51.170448
Templates	2015-08-31 11:56:01.315872	ntuser.dat.LOG1	2015-09-01 13:37:51.155094
TEMPLA~1	2015-08-31 11:56:01.315872	ntuser.ini	2015-09-01 13:37:51.155094

Fourth location - \[root]\Users\gib5on\AppData\Roaming\Microsoft\Windows\$I30

Fifth location - \[root]\Users\gib5on\Desktop\$I30

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

```
FILENAME, PHYSICAL SIZE, LOGICAL SIZE, MODIFIED TIME, ACCESSED TIME, CHANGED TIME, CREATED TIME

desktop.ini, 288, 282, 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

secret.txt, 64, 64, 2015-09-01 13:39:02.121397, 2015-09-01 13:39:03.150068, 2015-09-01 13:39:02.125263, 2015-09-01 13:39:02.120916’

secret.txt~, 40, 35, 2015-08-31 13:18:57.007290, 2015-09-01 13:37:55.497395, 2015-09-01 13:39:02.125278, 2015-08-31 13:18:30.846045

secret.txt~ (slack at 0x1a0), 40, 35, 2015-08-31 13:18:57.007290, 2015-09-01 13:37:55.497395, 2015-09-01 13:39:02.125237, 2015-08-31 13:18:30.846045’

secret.txt~ (slack at 0x220), 40, 35, 2015-08-31 13:18:57.007290, 2015-09-01 13:37:55.497395, 2015-09-01 13:39:02.125237, 2015-08-31 13:18:30.846045’
```

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 LiberOffice 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/

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:\AF\ADD>ADD_User_Console.exe
Usage:
        /file filename
        /proc procname pid ppid createtimeL createTimeH exitTimeL exitTimeH
        /tcpCon localIP localPort remoteIP remotePort state
                Note that IP address 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

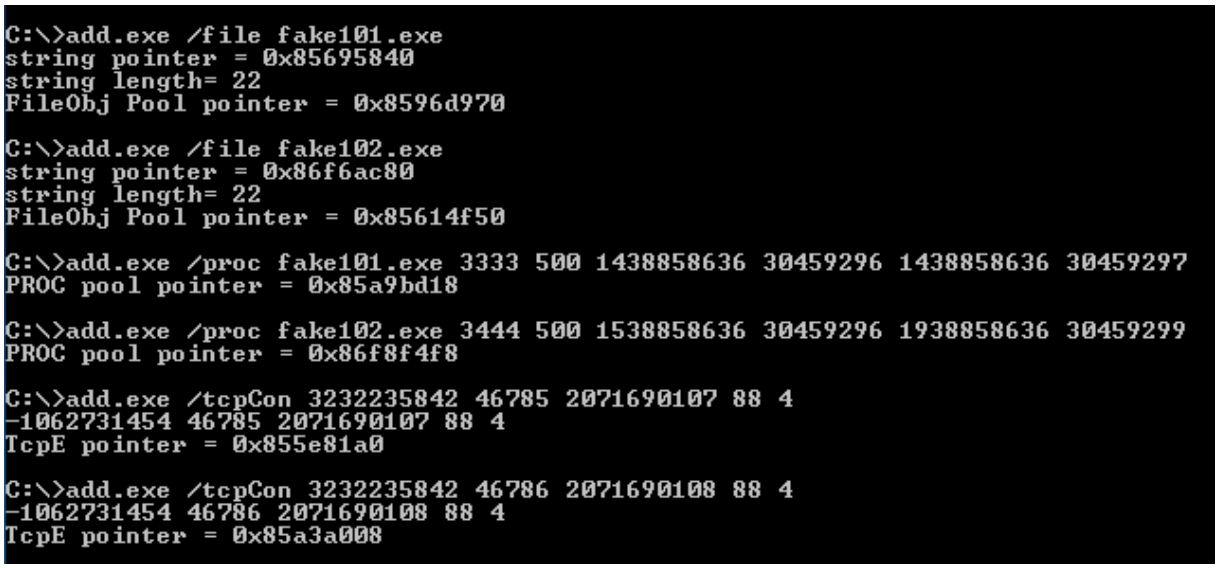
Memory image acquisition tools:

- 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 123.123.123.123:88 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):

Name	Pid	PPid	Thds	Hnds	Time
0x85e95030:wininit.exe	404	304	7	93	2015-08-17 06:40:19 UTC+0000
. 0x85f97d28:lsm.exe	516	404	12	156	2015-08-17 06:40:19 UTC+0000
. 0x85f63030:services.exe	500	404	21	253	2015-08-17 06:40:19 UTC+0000
.. 0x870481e0:taskhost.exe	1536	500	12	227	2015-08-17 06:40:20 UTC+0000
.. 0x86c91b90:VSSVC.exe	2408	500	6	123	2015-08-17 06:40:24 UTC+0000
.. 0x85f76830:svchost.exe	780	500	22	436	2015-08-17 06:40:19 UTC+0000
... 0x85ec3318:audiodg.exe	964	780	6	133	2015-08-17 06:40:19 UTC+0000

```
.. 0x852132c0:msdtc.exe        2192    500     16      157     2015-08-17 06:40:22 UTC+0000
.. 0x86fb2d28:svchost.exe      1304    500     25      330     2015-08-17 06:40:20 UTC+0000
.. 0x86d07308:SearchIndexer    2528    500     14      623     2015-08-17 06:40:27 UTC+0000
... 0x86d32520:SearchProtocol 2616    2528    7       260     2015-08-17 06:40:27 UTC+0000
... 0x86d354b0:SearchFilterHo 2636    2528    5       86      2015-08-17 06:40:27 UTC+0000
.. 0x87027030:svchost.exe      1432    500     14      306     2015-08-17 06:40:20 UTC+0000
.. 0x86d72030:sppsvc.exe       2844    500     7       179     2015-08-17 06:40:36 UTC+0000
.. 0x85eee4e0:svchost.exe      688     500     12      314     2015-08-17 06:40:19 UTC+0000
.. 0x851eed28:dllhost.exe      2084    500     18      210     2015-08-17 06:40:22 UTC+0000
.. 0x870a49c8:svchost.exe      956     500     7       98      2015-08-17 06:40:22 UTC+0000
.. 0x85e7f3d8:svchost.exe      816     500     23      415     2015-08-17 06:40:19 UTC+0000
... 0x8709baa8:dwm.exe         1664    816     7       121     2015-08-17 06:40:20 UTC+0000
.. 0x86c53030:TPAutoConnSvc.   628     500     11      146     2015-08-17 06:40:22 UTC+0000
... 0x86c8d850:TPAutoConnect. 2304    628     5       127     2015-08-17 06:40:23 UTC+0000
.. 0x85f0b6c0:svchost.exe      884     500     45      804     2015-08-17 06:40:19 UTC+0000
... 0x85cf36b8:taskeng.exe     3536    884     6       89      2015-08-17 06:40:51 UTC+0000
.. 0x870709a8:vmtoolsd.exe     1616    500     11      305     2015-08-17 06:40:20 UTC+0000
.. 0x85f5e508:svchost.exe      856     500     19      604     2015-08-17 06:40:19 UTC+0000
.. 0x86f31d28:svchost.exe      1116    500     21      387     2015-08-17 06:40:20 UTC+0000
.. 0x85ef7510:svchost.exe      992     500     7       120     2015-08-17 06:40:20 UTC+0000
.. 0x86c53d28:dllhost.exe      1968    500     22      203     2015-08-17 06:40:22 UTC+0000
.. 0x85e97d28:svchost.exe      612     500     16      376     2015-08-17 06:40:19 UTC+0000
... 0x8718fa40:WmiPrvSE.exe    768     612     10      143     2015-08-17 06:40:21 UTC+0000
... 0x8715a340:WmiPrvSE.exe    200     612     8       117     2015-08-17 06:40:21 UTC+0000
.. 0x86f91030:spoolsv.exe      1256    500     18      385     2015-08-17 06:40:20 UTC+0000
. 0x85f02540:lsass.exe         508     404     10      640     2015-08-17 06:40:19 UTC+0000
 0x85f79d28:csrss.exe          316     304     9       562     2015-08-17 06:40:19 UTC+0000
 0x851496f0:System             4       0       81      501     2015-08-17 06:40:17 UTC+0000
. 0x85d37200:smss.exe          232     4       4       31      2015-08-17 06:40:17 UTC+0000
 0x870abd28:explorer.exe       1724    1636    36      898     2015-08-17 06:40:20 UTC+0000
. 0x85cf59e8:DumpIt.exe        3488    1724    2       39      2015-08-17 06:40:50 UTC+0000
. 0x8717bc00:vmtoolsd.exe      352     1724    8       187     2015-08-17 06:40:21 UTC+0000
 0x85eab030:csrss.exe          396     388     10      254     2015-08-17 06:40:19 UTC+0000
. 0x86dfd450:conhost.exe       3500    396     2       51      2015-08-17 06:40:50 UTC+0000
. 0x86c64030:conhost.exe       2316    396     1       33      2015-08-17 06:40:23 UTC+0000
 0x85eff030:winlogon.exe       460     388     7       128     2015-08-17 06:40:19 UTC+0000
 0x853e0030:GWX.exe            2812    2800    7       211     2015-08-17 06:40:36 UTC+0000
```

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

```
Name                           Pid     PPid    Thds    Hnds    Time
----                           ---     ----    ----    ----    -----
 0x85ec9d28:wininit.exe        396     304     3       79      2015-08-17 06:42:50 UTC+0000
. 0x85f7c368:lsass.exe         512     396     9       571     2015-08-17 06:42:50 UTC+0000
. 0x85e93af0:lsm.exe           520     396     11      146     2015-08-17 06:42:50 UTC+0000
. 0x85eca658:services.exe      496     396     7       211     2015-08-17 06:42:50 UTC+0000
.. 0x86f14200:svchost.exe      1004    496     6       112     2015-08-17 06:42:51 UTC+0000
.. 0x85d303e8:svchost.exe      272     496     6       95      2015-08-17 06:42:53 UTC+0000
.. 0x872ed550:SearchIndexer    2680    496     11      607     2015-08-17 06:42:58 UTC+0000
... 0x87344030:SearchFilterHo 2168    2680    5       557     2015-08-17 06:48:49 UTC+0000
... 0x854c8030:SearchProtocol 4084    2680    7       218     2015-08-17 06:48:49 UTC+0000
.. 0x86fdfd28:svchost.exe      1304    496     18      310     2015-08-17 06:42:51 UTC+0000
.. 0x85f46d28:svchost.exe      700     496     8       288     2015-08-17 06:42:50 UTC+0000
.. 0x8708f838:taskhost.exe     1584    496     9       223     2015-08-17 06:42:51 UTC+0000
.. 0x8737c9a0:sppsvc.exe       3124    496     4       165     2015-08-17 06:43:07 UTC+0000
.. 0x85fceab8:svchost.exe      828     496     19      404     2015-08-17 06:42:50 UTC+0000
... 0x870c5148:dwm.exe         1704    828     5       122     2015-08-17 06:42:51 UTC+0000
.. 0x8704ed28:vmtoolsd.exe     1472    496     8       297     2015-08-17 06:42:51 UTC+0000
.. 0x87023768:svchost.exe      1420    496     10      278     2015-08-17 06:42:51 UTC+0000
.. 0x85216588:dllhost.exe      2128    496     16      208     2015-08-17 06:42:53 UTC+0000
.. 0x87090820:msdtc.exe        2232    496     14      154     2015-08-17 06:42:54 UTC+0000
.. 0x872f8200:TrustedInstall  2908    496     5       120     2015-08-17 06:45:39 UTC+0000
.. 0x85ea0a40:svchost.exe      868     496     13      545     2015-08-17 06:42:50 UTC+0000
.. 0x870e1030:svchost.exe      2412    496     14      377     2015-08-17 06:44:53 UTC+0000
.. 0x86fc28c8:spoolsv.exe      1276    496     13      357     2015-08-17 06:42:51 UTC+0000
.. 0x85eb6d28:svchost.exe      620     496     12      366     2015-08-17 06:42:50 UTC+0000
... 0x8714dd28:WmiPrvSE.exe    2032    620     5       118     2015-08-17 06:42:52 UTC+0000
... 0x8713da60:dllhost.exe     2484    620     7       81      2015-08-17 06:48:45 UTC+0000
.. 0x8717ed28:TPAutoConnSvc    1768    496     9       142     2015-08-17 06:42:53 UTC+0000
... 0x87236970:TPAutoConnect  2400    1768    5       126     2015-08-17 06:42:54 UTC+0000
.. 0x86f76758:svchost.exe      1128    496     15      389     2015-08-17 06:42:51 UTC+0000
.. 0x85ef33e8:svchost.exe      788     496     20      463     2015-08-17 06:42:50 UTC+0000
... 0x85f70418:audiodg.exe     968     788     6       133     2015-08-17 06:42:50 UTC+0000
.. 0x85ec1590:svchost.exe      892     496     32      1189    2015-08-17 06:42:50 UTC+0000
```

```
 0x85f62230:csrss.exe          312    304    9      452    2015-08-17 06:42:50 UTC+0000
 0x85ee6030:csrss.exe          404    384    10     232    2015-08-17 06:42:50 UTC+0000
. 0x872394a8:conhost.exe       2408   404    1      33     2015-08-17 06:42:54 UTC+0000
. 0x870b8d28:conhost.exe       3504   404    2      52     2015-08-17 06:43:20 UTC+0000
. 0x87313490:conhost.exe       1936   404    2      52     2015-08-17 06:48:46 UTC+0000
 0x85f4b030:winlogon.exe       460    384    5      126    2015-08-17 06:42:50 UTC+0000
 0x851496f0:System             4      0      82     521    2015-08-17 06:42:48 UTC+0000
. 0x85d35508:smss.exe          232    4      2      30     2015-08-17 06:42:48 UTC+0000
 0x870f7a40:explorer.exe       1836   1668   33     932    2015-08-17 06:42:52 UTC+0000
. 0x873b5900:cmd.exe           3496   1836   1      22     2015-08-17 06:43:20 UTC+0000
. 0x87241300:DumpIt.exe        3160   1836   2      39     2015-08-17 06:48:46 UTC+0000
. 0x85e73d28:vmtoolsd.exe      888    1836   7      196    2015-08-17 06:42:52 UTC+0000
 0x87276a30:GWX.exe            3088   3076   4      196    2015-08-17 06:43:07 UTC+0000
```

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

```
Offset(P)             Name           PID    PPID   PDB            Time created
       Time exited
```

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

```
Offset(P)             Name           PID    PPID   PDB            Time created
       Time exited
0x000000007dcdb458    fake101.exe    3333   500    0x12345680     2015-07-23 23:36:01
       2015-07-23 23:43:11
0x000000007de4e848    fake102.exe    3444   500    0x12345680     2015-07-23 23:36:11
       2015-07-23 23:58:20
```

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

```
Offset(P) Name                 PID    pslist psscan thrdproc pspcid csrss session deskthrd
      ExitTime
0x7df7bc00 vmtoolsd.exe        352    True False True True True True True
0x7df5a340 WmiPrvSE.exe        200    True False True True True True True
0x7de709a8 vmtoolsd.exe        1616   True False True True True True True
0x7e307308 SearchIndexer       2528   True False True True True True True
0x7f095030 wininit.exe         404    True False True True True True True
0x7f0f7510 svchost.exe         992    True False True True True True True
0x7f07f3d8 svchost.exe         816    True False True True True True True
0x7f0c3318 audiodg.exe         964    True False True True True True True
0x7e253030 TPAutoConnSvc       628    True False True True True True True
0x7fc132c0 msdtc.exe           2192   True False True True True True True
0x7e3fd450 conhost.exe         500    True False True True True True True
0x7e1b2d28 svchost.exe         1304   True False True True True True True
0x7e332520 SearchProtocol      2616   True False True True True True True
0x7de27030 svchost.exe         1432   True False True True True True True
0x7de9baa8 dwm.exe             1664   True False True True True True True
0x7e372030 sppsvc.exe          2844   True False True True True True True
0x7f097d28 svchost.exe         612    True False True True True True True
0x7f163030 services.exe        500    True False True True True True False
0x7f15e508 svchost.exe         856    True False True True True True True
0x7f2f36b8 taskeng.exe         3536   True False True True True True True
0x7f10b6c0 svchost.exe         884    True False True True True True True
0x7f102540 lsass.exe           508    True False True True True True False
0x7de481e0 taskhost.exe        1536   True False True True True True True
0x7fde0030 GWX.exe             2812   True False True True True True True
0x7dea49c8 svchost.exe         956    True False True True True True True
0x7e3354b0 SearchFilterHo      2636   True False True True True True True
0x7deabd28 explorer.exe        1724   True False True True True True True
0x7e28d850 TPAutoConnect       2304   True False True True True True True
0x7f0ff030 winlogon.exe        460    True False True True True True True
0x7ff6dd28 dllhost.exe         2084   True False True True True True True
0x7e264030 conhost.exe         2316   True False True True True True True
0x7df8fa40 WmiPrvSE.exe        768    True False True True True True True
0x7e191030 spoolsv.exe         1256   True False True True True True True
0x7f0ee4e0 svchost.exe         688    True False True True True True True
0x7e291b90 VSSVC.exe           2408   True False True True True True True
0x7e131d28 svchost.exe         1116   True False True True True True True
0x7f176830 svchost.exe         780    True False True True True True True
```

```
0x7f2f59e8 DumpIt.exe         3488    True False True True True True True
0x7e253d28 dllhost.exe        1968    True False True True True True True
0x7f197d28 lsm.exe            516     True False True True True True False
0x7f337200 smss.exe           232     True False True True False False False
0x7f179d28 csrss.exe          316     True False True True False True True
0x7f0ab030 csrss.exe          396     True False True True False True True
0x7ffc86f0 System             4       True False True True False False False
0x7e011030 dllhost.exe        3416    False False False False False False True
       2015-08-17 06:40:55 UTC+0000
```

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

```
Offset(P).Name                PID     pslist psscan thrdproc pspcid csrss session deskthrd
       ExitTime
0x7dcf8200.TrustedInstall     2908    True False True True True True False
0x7e114200.svchost.exe        1004    True False True True True True False
0x7fc16588.dllhost.exe        2128    True False True True True True False
0x7dc36970.TPAutoConnect      2400    True False True True True True True
0x7f0c1590.svchost.exe        892     True False True True True True False
0x7f0ca658.services.exe       496     True False True True True True False
0x7df7ed28.TPAutoConnSvc      1768    True False True True True True False
0x7f170418 audiodg.exe        968     True False True True True True True
0x7e1dfd28 svchost.exe        1304    True False True True True True True
0x7dc41300 DumpIt.exe         3160    True False True True True True True
0x7df4dd28 WmiPrvSE.exe       2032    True False True True True True False
0x7dd7c9a0 sppsvc.exe         3124    True False True True True True True
0x7fac8030 SearchProtocol     4084    True False True False True True False
0x7f14b030 winlogon.exe       460     True False True True True True True
0x7f0c9d28 wininit.exe        396     True False True True True True True
0x7dc76a30 GWX.exe            3088    True False True True True True True
0x7dee1030 svchost.exe        2412    True False True True True True False
0x7dec5148 dwm.exe            1704    True False True True True True True
0x7f146d28 svchost.exe        700     True False True True True True True
0x7f1ceab8 svchost.exe        828     True False True True True True False
0x7f0a0a40 svchost.exe        868     True False True True True True True
0x7de90820 msdtc.exe          2232    True False True True True True True
0x7e1c28c8 spoolsv.exe        1276    True False True True True True True
0x7ddb5900 cmd.exe            3496    True False True True True True True
0x7dced550 SearchIndexer      2680    True False True True True True False
0x7de8f838 taskhost.exe       1584    True False True True True True True
0x7def7a40 explorer.exe       1836    True False True True True True True
0x7f0f33e8 svchost.exe        788     True False True True True True True
0x7de4ed28 vmtoolsd.exe       1472    True False True True True True False
0x7e176758 svchost.exe        1128    True False True True True True True
0x7dc394a8 conhost.exe        2408    True False True True True True True
0x7df3da60 dllhost.exe        2484    True False True True True True False
0x7dd13490 conhost.exe        1936    True False True True True True True
0x7de23768 svchost.exe        1420    True False True True True True False
0x7f3303e8 svchost.exe        272     True False True True True True True
0x7f073d28 vmtoolsd.exe       888     True False True True True True True
0x7f093af0 lsm.exe            520     True False True True True True False
0x7f17c368 lsass.exe          512     True False True True True True False
0x7f0b6d28 svchost.exe        620     True False True True True True False
0x7deb8d28 conhost.exe        3504    True False True True True True True
0x7f335508 smss.exe           232     True False True True False False False
0x7f0e6030 csrss.exe          404     True False True True False True True
0x7f162230 csrss.exe          312     True False True True False True True
0x7ffc86f0 System             4       True False True True False False False
0x7f168540 dllhost.exe        316     False False False False False False True
       2015-08-17 06:48:50 UTC+0000
0x7dd44030 SearchFilterHo     2168    True False True False False True False
0x7dcdb458 fake101.exe        3333    False True False False False False False
       2015-07-23 23:43:11 UTC+0000
0x7de4e848 fake102.exe        3444    False True False False False False False
       2015-07-23 23:58:20 UTC+0000
```

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

Offset(P)	Proto	Local Address	Foreign Address	State	Pid	Owner
0x7df58cf8	TCPv4	0.0.0.0:445	0.0.0.0:0	LISTENING	4	System
0x7e1be6e0	TCPv4	0.0.0.0:49155	0.0.0.0:0	LISTENING	500	services.exe
0x7e27c9b0	TCPv4	192.168.117.135:139	0.0.0.0:0	LISTENING	4	System

0x7e362d48	TCPv4	0.0.0.0:49156	0.0.0.0:0	LISTENING	508	lsass.exe
0x7e362ea8	TCPv4	0.0.0.0:49156	0.0.0.0:0	LISTENING	508	lsass.exe
0x7e02f008	TCPv4	192.168.117.135:49159	207.46.194.14:80	CLOSED		
0x7e03c008	TCPv4	192.168.117.135:49160	207.46.194.14:80	CLOSED		
0x7e04d740	TCPv4	192.168.117.135:49162	184.86.2.152:443	CLOSED		
0x7e04d950	TCPv4	192.168.117.135:49161	184.86.2.152:443	CLOSED		
0x7e05d618	TCPv4	192.168.117.135:49168	23.37.37.163:80	CLOSED		
0x7e05e008	TCPv4	192.168.117.135:49167	108.162.232.201:80	CLOSED		
0x7e16c600	TCPv4	192.168.117.135:49164	207.46.194.25:443	CLOSED		
0x7e2cbc30	TCPv4	192.168.117.135:49157	191.232.139.253:443	ESTABLISHED		
0x7f072f18	TCPv4	0.0.0.0:49153	0.0.0.0:0	LISTENING	780	svchost.exe
0x7f0855d8	TCPv4	0.0.0.0:49154	0.0.0.0:0	LISTENING	884	svchost.exe
0x7f08beb8	TCPv4	0.0.0.0:49155	0.0.0.0:0	LISTENING	500	services.exe
0x7f09f908	TCPv4	0.0.0.0:49153	0.0.0.0:0	LISTENING	780	svchost.exe
0x7f107348	TCPv4	0.0.0.0:49154	0.0.0.0:0	LISTENING	884	svchost.exe
0x7f108b78	TCPv4	0.0.0.0:135	0.0.0.0:0	LISTENING	688	svchost.exe
0x7f12f4d8	TCPv4	0.0.0.0:49152	0.0.0.0:0	LISTENING	404	wininit.exe
0x7f15a4a0	TCPv4	0.0.0.0:49152	0.0.0.0:0	LISTENING	404	wininit.exe
0x7fa49008	TCPv4	192.168.117.135:49165	8.27.13.125:443	CLOSED		

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

Offset(P)	Proto	Local Address	Foreign Address	State	Pid	Owner
0x7dd6d260	TCPv4	0.0.0.0:49156	0.0.0.0:0	LISTENING	512	lsass.exe
0x7dd6f8f8	TCPv4	0.0.0.0:49156	0.0.0.0:0	LISTENING	512	lsass.exe
0x7df31230	TCPv4	0.0.0.0:445	0.0.0.0:0	LISTENING	4	System
0x7dfcb378	TCPv4	0.0.0.0:49155	0.0.0.0:0	LISTENING	496	services.exe
0x7dfcb648	TCPv4	0.0.0.0:49155	0.0.0.0:0	LISTENING	496	services.exe
0x7e19d5a8	TCPv4	0.0.0.0:49154	0.0.0.0:0	LISTENING	892	svchost.exe
0x7dde6cd0	TCPv4	192.168.1.66:46786	123.123.123.124:88	ESTABLISHED	0	
0x7f0af4d0	TCPv4	0.0.0.0:49153	0.0.0.0:0	LISTENING	788	svchost.exe
0x7f121ea8	TCPv4	0.0.0.0:49154	0.0.0.0:0	LISTENING	892	svchost.exe
0x7f15dea0	TCPv4	192.168.117.135:139	0.0.0.0:0	LISTENING	4	System
0x7f163ca8	TCPv4	0.0.0.0:49152	0.0.0.0:0	LISTENING	396	wininit.exe
0x7f165898	TCPv4	0.0.0.0:135	0.0.0.0:0	LISTENING	700	svchost.exe
0x7f175758	TCPv4	0.0.0.0:135	0.0.0.0:0	LISTENING	700	svchost.exe
0x7f1a6588	TCPv4	0.0.0.0:49152	0.0.0.0:0	LISTENING	396	wininit.exe
0x7f1ced98	TCPv4	0.0.0.0:49153	0.0.0.0:0	LISTENING	788	svchost.exe
0x7fae5008	TCPv4	192.168.1.66:46785	123.123.123.123:88	ESTABLISHED	0	

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 build 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 legitemate 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=0x000000007dcdb458

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=0x000000007dcdb458
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=0x000000007dcdb458

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=0x000000007dcdb458 --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=0x000000007dcdb458 --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*.

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

Offset(P)          Name                Base                Size      File
------------------ ------------------- ---------- ---------- ----
0x000000007dd14ec8  asyncmac.sys        0xa0925000          0x9000    \SystemRoot\system32\DRIVERS\asyncmac.sys
0x000000007de02d90  mrxsmb.sys          0x97cc0000          0x23000   \SystemRoot\system32\DRIVERS\mrxsmb.sys
0x000000007de08188  mrxsmb20.sys        0x97d1f000          0x1c000   \SystemRoot\system32\DRIVERS\mrxsmb20.sys
0x000000007deb01e8  srv.sys             0xa0869000          0x52000   \SystemRoot\System32\DRIVERS\srv.sys
.
.
.
0x007f0fb688        DumpIt.sys          0xa0936000          0xc0      \??\C:\Windows\system32\Drivers\DumpIt.sys
0x000000007f105330  drmk.sys            0x9368b000          0x19000   \SystemRoot\system32\drivers\drmk.sys
0x000000007f2517f0  crashdmp.sys        0x936a4000          0xd000    \SystemRoot\System32\Drivers\crashdmp.sys
.
.
.
0x000000007fa62e38  spsys.sys           0xa08bb000          0x6a00    \SystemRoot\system32\drivers\spsys.sys
0x000000007fa65a48  add.sys             0xa092e000          0x8000    \??\C:\AF\ADD\add.sys
```

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*.

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

Offset: 0x7c09c0
Order: 4
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: add
Display Name: add
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_RUNNING
Binary Path: \Driver\add
```

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*.

```
**************************************************
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: 0x3ba500 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: crl
Cmd #6 @ 0x3b8580: cls
Cmd #7 @ 0x3ba668: add.exe /file fake101.exe
Cmd #8 @ 0x3bee90: 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
```

```
Cmd #11 @ 0x3be878: add.exe /tcpCon 3232235842 46785 2071690107 88 4
Cmd #12 @ 0x3be8e8: add.exe /tcpCon 3232235842 46786 2071690108 88 4
Cmd #22 @ 0xff818488: ?
Cmd #25 @ 0xff818488: ?
Cmd #36 @ 0x3800c4: ;?;?8???8
Cmd #37 @ 0x3b70e8: ;?8????
**********************************************
CommandProcess: conhost.exe Pid: 1936
CommandHistory: 0x359338 Application: DumpIt.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x60
Cmd #3 @ 0x340030: ??????????????????????????????????????????????????_
Cmd #17 @ 0x310039: ???_
Cmd #22 @ 0xff818488: ?
Cmd #25 @ 0xff818488: ?
Cmd #36 @ 0x3200c4: 5?5?2???2
Cmd #37 @ 0x355e58: 5?2????
```

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

2110633412 fake101.exe
7188502 ;add.exe /file fake101.exestring pointer = 0x870687c0string length= 22FileObj Pool
pointer = 0x873c9ee8C:\>add.exe /file fake102.exestring pointer = 0x86fa34d0string length=
22FileObj Pool pointer = 0x87283318C:\>add.exe /proc fake101.exe 3333 500 1438858636
30459296 1438858636 30459297PROC pool pointer = 0x872db430C:\>add.exe /proc fake102.exe 3444
500 1538858636 30459296 1938858636 30459299PROC pool pointer = 0x8704e820C:\>add.exe /tcpCon
3232235842 46785 2071690107 88 4-1062731454 46785 2071690107 88 4TcpE pointer =
0x854e5008C:\>add.exe /tcpCon 3232235842 46786 2071690108 88 4-1062731454 46786
2071690108 88 4TcpE pointer = 0x873e6cd0C:\>
1925063120 exe /proc fake101.exe 3333
2000262690 eadd.exe /file fake101.exe
2012776954 eadd.exe /file fake101.exe
2066118248 2add.exe /file fake101.exe
2072370898 add.exe /proc fake101.exe 3333 500 1438858636 30459296 1438858636 304592972
2072626776 C:\>add.exe /file fake101.exe
string pointer = 0x870687c0
string length= 22
FileObj Pool pointer = 0x873c9ee8
C:\>add.exe /file fake102.exe
string pointer = 0x86fa34d0
string length= 22
FileObj Pool pointer = 0x87283318
C:\>add.exe /proc fake101.exe 3333 500 1438858636 30459296 1438858636 30459297 PROC pool
pointer = 0x872db430
C:\>add.exe /proc fake102.exe 3444 500 1538858636 30459296 1938858636 30459299 PROC pool
pointer = 0x8704e820
2112260032 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

2000262684 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.exestring pointer = 0x870687c0string length= 22FileObj Pool
pointer = 0x873c9ee8C:\>add.exe /file fake102.exestring pointer = 0x86fa34d0string length=
22FileObj Pool pointer = 0x87283318C:\>add.exe /proc fake101.exe 3333 500 1438858636
30459296 1438858636 30459297PROC pool pointer = 0x872db430C:\>add.exe /proc fake102.exe 3444
500 1538858636 30459296 1938858636 30459299PROC pool pointer = 0x8704e820C:\>add.exe /tcpCon
3232235842 46785 2071690107 88 4-1062731454 46785 2071690107 88 4TcpE pointer =
0x854e5008C:\>add.exe /tcpCon 3232235842 46786 2071690108 88 4-1062731454 46786 2071690108
88 4TcpE pointer = 0x873e6cd0C:\>
 15025644 C:\add.exe
 15025668 add.exe /proc fake102.exe 3444 500 1538858636 30459296 1938858636 30459299
 15025820 add.exe /proc fake102.exe 3444 500 1538858636 30459296 1938858636 30459299
 16057880 C:\add.exe
 16058414 Jadd.exe /tcpCon 3232235842 46786 2071690108 88 4
 121129728 \Device\HarddiskVolume1\add.exe
 156099632 \Device\HarddiskVolume1\add.exe:Zone.Identifier
 203505646 - add.exe
 216549258 add.exe
 228081920 \add.exe
 377807300 C:\add.exe
 387909874 add.exe
 449489936 \??\C:\add.exe
 497480968 \Device\HarddiskVolume1\add.exe
 531113536 add.exe /tcpCon 3232235842 46786 2071690108 88 4
 553245704 C:\add.exe
 744695056 \??\C:\add.exe
 746308880 \??\C:\add.exe
 760722866 add.exe
 852455432 \add.exe
1609042694 add.exe
1857051920 C:\add.exe
1956092888 \add.exe:Zone.Identifier
1993060068 \Device\HarddiskVolume1\add.exe
2000262690 eadd.exe /file fake101.exe
2008028348 \Device\HarddiskVolume1\add.exe
2008030876 \Device\HarddiskVolume1\add.exe
2011061434 eadd.exe
2011453268 \add.exe
2011453348 \add.exe
2011453444 \add.exe
2011453524 \add.exe
2012776954 eadd.exe /file fake101.exe
2016965348 \Device\HarddiskVolume1\add.exe
2016968236 \Device\HarddiskVolume1\add.exe
2022553788 \Device\HarddiskVolume1\add.exe
2023762538 eadd.exe
2051568254 Jadd.exe /tcpCon 3232235842 46786 2071690108 88 4
2060744282 =C:\add.exe
2060744306 ladd.exe /tcpCon 3232235842 46786 2071690108 88 4
2060744408 add.exe /tcpCon 3232235842 46786 2071690108 88 4
2066118248 2add.exe /file fake101.exe
2066724024 add.exe
2069369142 Jadd.exe
2069963472 add.exe
2069963522 add.exe
2071048192 C:\>add.exe /tcpCon 3232235842 46785 2071690107 88 4 -1062731454 46785 2071690107
88 4 TcpE pointer = 0x854e5008 C:\>add.exe /tcpCon 3232235842 46786 2071690108 88 4 -
1062731454 46786 2071690108 88 4 TcpE pointer = 0x873e6cd0 C:\>
2072369272 `add.exe /tcpCon 3232235842 46785 2071690107 88 4
2072369384 `add.exe /tcpCon 3232235842 46786 2071690108 88 45 2
2072369502 ;and Prompt - add.exe /tcpCon 3232235842 46786 2071690108 88 4
2072370832 2add.exe /file fake102.exeex
2072370898 add.exe /proc fake101.exe 3333 500 1438858636 30459296 1438858636 304592972
2072371058 add.exe /proc fake102.exe 3444 500 1538858636 30459296 1938858636 30459
2072626776 C:\>add.exe /file fake101.exe string pointer = 0x870687c0 string length= 22
FileObj Pool pointer = 0x873c9ee8 C:\>add.exe /file fake102.exe string pointer = 0x86fa34d0
```

```
string length= 22 FileObj Pool pointer = 0x87283318 C:\>add.exe /proc fake101.exe 3333 500
1438858636 30459296 1438858636 30459297 PROC pool pointer = 0x872db430 C:\>add.exe /proc
fake102.exe 3444 500 1538858636 30459296 1938858636 30459299 PROC pool pointer = 0x8704e820
2074918054 add.exe
2074918614 IC:\add.exe
2075934576 \\?\C:\add.exe:Zone.Identifier
2107654316 \Device\HarddiskVolume1\add.exe
2107654488 \Device\HarddiskVolume1\add.exe
2107657472 \Device\HarddiskVolume1\add.exe
2109998900 evice\harddiskvolume1\add.exe
2140368058 eadd.exe
```

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/

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.

```
C:\AF\Dementia>Dementia.exe
Dementia - v1.0 -- Windows memory anti-forensics suite
Copyright (c) 2012, Luka Milkovic <milkovic.luka@gmail.com> or <luka.milkovic@infigo.hr>

Usage: Dementia.exe [-h|--help -d|--debug -f|--file] -m <evasion_method> [-a|--args]
General options:
  -h [ --help ]          view program description
  -d [ --debug ]         print verbose/detailed program output - useful for
                         debugging
  -f [ --file ]          write all program output to "log.txt" file
  -m [ --method ] arg    Evasion method. Following methods are supported
                         (specify wanted method by number):

                         1 - Memoryze usermode evasion: Hides kernel level
                                                        objects - process block,
                                                        threads and network
                                                        connections from the
                                                        usermode
                         2 - Generic kernel-mode evasion module: Hides kernel
                                                                 level objects
                                                                 related to
                                                                 target process
                                                                 using kernel
                                                                 driver
                         3 - Generic kernel-mode evasion module: Hides kernel
                                                                 level objects
                                                                 related to
                                                                 target process
                                                                 using kernel
                                                                 driver

  -a [ --args ] arg      pass specific arguments to the evasion method (use info
                         for method specific help) -- arguments are passed in
                         quotes!
```

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

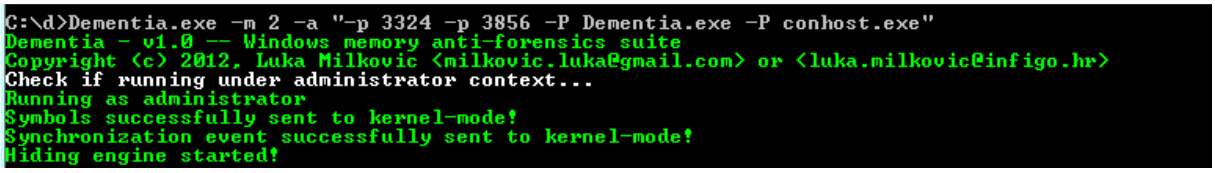
Memory image acquisition tools:

- 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



Dementia – testing scenario

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

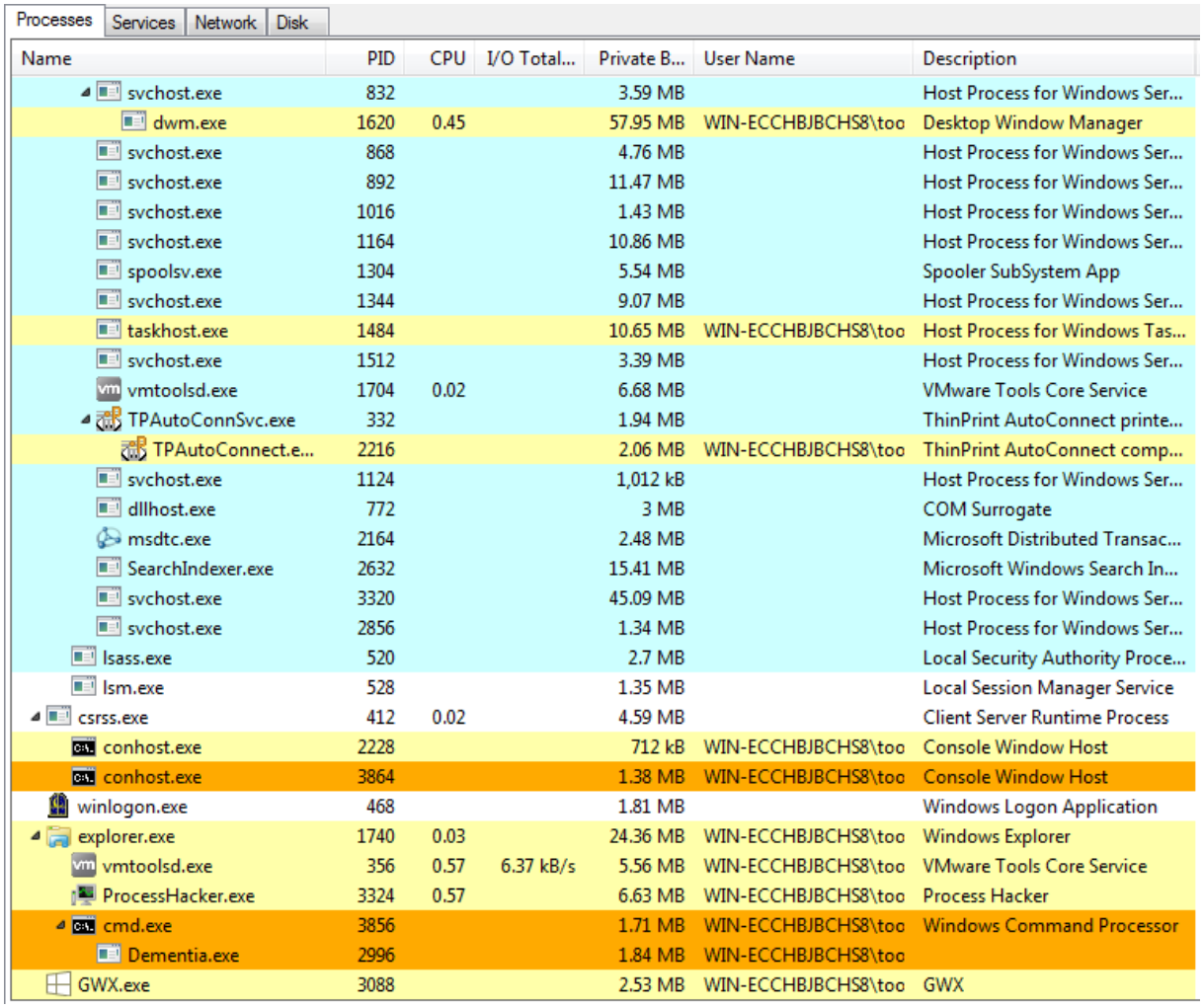
The main goal for the testing scenario was to observe how the tool works and whether it supports the most popular memory image acquisition tools. We tested five different operating systems. The 'kernel mode hiding based on hooks' method was used on x86 architecture and kernel-mode hiding based on file 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` and `PFN Mapping`.. All of them presented the same output:

pstree – list all active processes (before using Dementia)

Name	Pid	PPid	Thds	Hnds	Time
0x85222030:explorer.exe	1764	1700	34	880	2015-08-21 11:40:18 UTC+0000
. 0x86aaad28:FTK Imager.exe	3968	1764	20	392	2015-08-21 11:40:58 UTC+0000
. 0x870f92a0:vmtoolsd.exe	1412	1764	8	181	2015-08-21 11:40:19 UTC+0000
. 0x86c3ed28:ProcessHacker	2920	1764	9	282	2015-08-21 11:40:39 UTC+0000
0x86a71030:wininit.exe	404	296	7	88	2015-08-21 11:40:16 UTC+0000
. 0x86dd1030:lsm.exe	524	404	11	152	2015-08-21 11:40:17 UTC+0000
. 0x86dc56a0:lsass.exe	516	404	10	610	2015-08-21 11:40:17 UTC+0000
. 0x86db9d28:services.exe	508	404	19	239	2015-08-21 11:40:17 UTC+0000

```
.. 0x86f9b030:spoolsv.exe     1284    508     18      360     2015-08-21 11:40:17 UTC+0000
.. 0x86b0f7e8:VSSVC.exe       2440    508     6       121     2015-08-21 11:40:21 UTC+0000
.. 0x86e8d490:svchost.exe     876     508     44      775     2015-08-21 11:40:17 UTC+0000
... 0x86c9ad28:taskeng.exe    3116    876     6       88      2015-08-21 11:40:48 UTC+0000
.. 0x86e51778:svchost.exe     784     508     23      432     2015-08-21 11:40:17 UTC+0000
... 0x86ede518:audiodg.exe    976     784     6       132     2015-08-21 11:40:17 UTC+0000
.. 0x86f00b48:vmtoolsd.exe    1552    508     11      303     2015-08-21 11:40:18 UTC+0000
.. 0x870ab320:TPAutoConnSvc   240     508     10      146     2015-08-21 11:40:19 UTC+0000
... 0x86ac66a8:TPAutoConnect  2224    240     6       130     2015-08-21 11:40:20 UTC+0000
.. 0x86ee9030:svchost.exe     1316    508     25      320     2015-08-21 11:40:18 UTC+0000
.. 0x870de1f8:dllhost.exe     636     508     22      205     2015-08-21 11:40:19 UTC+0000
.. 0x86ecf8d8:svchost.exe     1460    508     13      291     2015-08-21 11:40:18 UTC+0000
.. 0x86e64d28:svchost.exe     824     508     24      416     2015-08-21 11:40:17 UTC+0000
... 0x85208030:dwm.exe        1716    824     7       124     2015-08-21 11:40:18 UTC+0000
.. 0x86ba5030:sppsvc.exe      2832    508     7       176     2015-08-21 11:40:34 UTC+0000
.. 0x86f04518:svchost.exe     1004    508     7       113     2015-08-21 11:40:17 UTC+0000
.. 0x86e77518:svchost.exe     848     508     19      609     2015-08-21 11:40:17 UTC+0000
.. 0x86df77d8:svchost.exe     696     508     13      297     2015-08-21 11:40:17 UTC+0000
.. 0x851e9030:taskhost.exe    1636    508     12      229     2015-08-21 11:40:18 UTC+0000
.. 0x86acd7d8:svchost.exe     2232    508     7       96      2015-08-21 11:40:20 UTC+0000
.. 0x86d9b308:msdtc.exe       2152    508     16      158     2015-08-21 11:40:20 UTC+0000
.. 0x86e107a8:svchost.exe     620     508     17      377     2015-08-21 11:40:17 UTC+0000
... 0x870d4d28:WmiPrvSE.exe   252     620     8       140     2015-08-21 11:40:19 UTC+0000
.. 0x86f43030:svchost.exe     1136    508     22      427     2015-08-21 11:40:17 UTC+0000
.. 0x85207910:dllhost.exe     372     508     19      216     2015-08-21 11:40:19 UTC+0000
.. 0x86472a18:SearchIndexer   2540    508     15      606     2015-08-21 11:40:25 UTC+0000
... 0x86b88748:SearchFilterHo 2636    2540    5       84      2015-08-21 11:40:25 UTC+0000
... 0x86ed39b0:SearchProtocol 2616    2540    7       259     2015-08-21 11:40:25 UTC+0000
 0x86112030:csrss.exe         312     296     9       546     2015-08-21 11:40:16 UTC+0000
 0x86a7ad28:csrss.exe         412     396     10      278     2015-08-21 11:40:16 UTC+0000
. 0x86ad7d28:conhost.exe      2244    412     1       33      2015-08-21 11:40:20 UTC+0000
 0x86da4d28:winlogon.exe      468     396     6       122     2015-08-21 11:40:17 UTC+0000
 0x86ba0c28:GWX.exe           2800    2788    6       209     2015-08-21 11:40:34 UTC+0000
 0x851496f0:System            4       0       82      494     2015-08-21 11:40:15 UTC+0000
0x85d639e0:smss.exe           2       28      4       431     2015-08-21 11:40:15 UTC+0000
```

To show the differences between acquired memory image and real system status, the screen shot below shows Process Hacker output just before image acquisition.



Processes | Services | Network | Disk

Name	PID	CPU	I/O Total...	Private B...	User Name	Description
svchost.exe	832			3.59 MB		Host Process for Windows Ser...
dwm.exe	1620	0.45		57.95 MB	WIN-ECCHBJBCHS8\too	Desktop Window Manager
svchost.exe	868			4.76 MB		Host Process for Windows Ser...
svchost.exe	892			11.47 MB		Host Process for Windows Ser...
svchost.exe	1016			1.43 MB		Host Process for Windows Ser...
svchost.exe	1164			10.86 MB		Host Process for Windows Ser...
spoolsv.exe	1304			5.54 MB		Spooler SubSystem App
svchost.exe	1344			9.07 MB		Host Process for Windows Ser...
taskhost.exe	1484			10.65 MB	WIN-ECCHBJBCHS8\too	Host Process for Windows Tas...
svchost.exe	1512			3.39 MB		Host Process for Windows Ser...
vmtoolsd.exe	1704	0.02		6.68 MB		VMware Tools Core Service
TPAutoConnSvc.exe	332			1.94 MB		ThinPrint AutoConnect printe...
TPAutoConnect.e...	2216			2.06 MB	WIN-ECCHBJBCHS8\too	ThinPrint AutoConnect comp...
svchost.exe	1124			1,012 kB		Host Process for Windows Ser...
dllhost.exe	772			3 MB		COM Surrogate
msdtc.exe	2164			2.48 MB		Microsoft Distributed Transac...
SearchIndexer.exe	2632			15.41 MB		Microsoft Windows Search In...
svchost.exe	3320			45.09 MB		Host Process for Windows Ser...
svchost.exe	2856			1.34 MB		Host Process for Windows Ser...
lsass.exe	520			2.7 MB		Local Security Authority Proce...
lsm.exe	528			1.35 MB		Local Session Manager Service
csrss.exe	412	0.02		4.59 MB		Client Server Runtime Process
conhost.exe	2228			712 kB	WIN-ECCHBJBCHS8\too	Console Window Host
conhost.exe	3864			1.38 MB	WIN-ECCHBJBCHS8\too	Console Window Host
winlogon.exe	468			1.81 MB		Windows Logon Application
explorer.exe	1740	0.03		24.36 MB	WIN-ECCHBJBCHS8\too	Windows Explorer
vmtoolsd.exe	356	0.57	6.37 kB/s	5.56 MB	WIN-ECCHBJBCHS8\too	VMware Tools Core Service
ProcessHacker.exe	3324	0.57		6.63 MB	WIN-ECCHBJBCHS8\too	Process Hacker
cmd.exe	3856			1.71 MB	WIN-ECCHBJBCHS8\too	Windows Command Processor
Dementia.exe	2996			1.84 MB	WIN-ECCHBJBCHS8\too	
GWX.exe	3088			2.53 MB	WIN-ECCHBJBCHS8\too	GWX

pstree – list all active processes (after using Dementia)

```
Name                                Pid     PPid    Thds    Hnds    Time
----                                ---     ----    ----    ----    -----
 0x87091518:                        0       0       0               1970-01-01 00:00:00 UTC+0000
. 0x851496f0:System                 4       0       81      503     2015-08-21 11:48:34 UTC+0000
.. 0x85dd3020:smss.exe              228     4       2       30      2015-08-21 11:48:34 UTC+0000
 0x851496f0:System                  4       0       81      503     2015-08-21 11:48:34 UTC+0000
 0x86a7c6a8:wininit.exe             404     296     4       80      2015-08-21 11:48:35 UTC+0000
. 0x86dde6b0:lsass.exe              520     404     8       571     2015-08-21 11:48:36 UTC+0000
. 0x86dc7030:lsm.exe                528     404     11      150     2015-08-21 11:48:36 UTC+0000
. 0x86da1770:services.exe           504     404     13      224     2015-08-21 11:48:35 UTC+0000
.. 0x86e3f9e0:svchost.exe           1344    504     21      326     2015-08-21 11:48:37 UTC+0000
.. 0x86f74d28:svchost.exe           1164    504     21      486     2015-08-21 11:48:36 UTC+0000
.. 0x86ea9030:spoolsv.exe           1304    504     14      326     2015-08-21 11:48:36 UTC+0000
.. 0x870a09f0:dllhost.exe           772     504     18      208     2015-08-21 11:48:38 UTC+0000
.. 0x86b35948:VSSVC.exe             2468    504     8       118     2015-08-21 11:48:40 UTC+0000
.. 0x86e62d28:vmtoolsd.exe          1704    504     11      298     2015-08-21 11:48:37 UTC+0000
.. 0x8709c520:dllhost.exe           1584    504     23      202     2015-08-21 11:48:38 UTC+0000
.. 0x85463a48:sppsvc.exe            3124    504     5       168     2015-08-21 11:48:53 UTC+0000
.. 0x86e2a030:svchost.exe           700     504     8       288     2015-08-21 11:48:36 UTC+0000
.. 0x8704c948:TPAutoConnSvc         332     504     11      141     2015-08-21 11:48:38 UTC+0000
... 0x86e754a0:TPAutoConnect        2216    332     6       123     2015-08-21 11:48:39 UTC+0000
.. 0x86ed0518:svchost.exe           832     504     23      408     2015-08-21 11:48:36 UTC+0000
... 0x86ffc478:dwm.exe              1620    832     6       121     2015-08-21 11:48:37 UTC+0000
.. 0x86b959b0:SearchIndexer         2632    504     14      587     2015-08-21 11:48:44 UTC+0000
... 0x86bdf788:SearchFilterHo       2724    2632    4       78      2015-08-21 11:48:44 UTC+0000
... 0x86bd8b00:SearchProtocol       2704    2632    8       261     2015-08-21 11:48:44 UTC+0000
.. 0x86ecdd28:taskhost.exe          1484    504     12      243     2015-08-21 11:48:37 UTC+0000
.. 0x86efa650:svchost.exe           868     504     15      561     2015-08-21 11:48:36 UTC+0000
.. 0x86e7d518:svchost.exe           784     504     21      411     2015-08-21 11:48:36 UTC+0000
... 0x86f19600:audiodg.exe          976     784     7       132     2015-08-21 11:48:36 UTC+0000
.. 0x87086518:svchost.exe           1124    504     7       94      2015-08-21 11:48:38 UTC+0000
```

```
.. 0x86ec6bb8:svchost.exe      1512   504    12     265    2015-08-21 11:48:37 UTC+0000
.. 0x86e09100:svchost.exe      624    504    13     368    2015-08-21 11:48:36 UTC+0000
.. 0x87171900:msdtc.exe        2164   504    15     153    2015-08-21 11:48:39 UTC+0000
.. 0x86f34c70:svchost.exe      1016   504    6      111    2015-08-21 11:48:36 UTC+0000
.. 0x86f0aba8:svchost.exe      892    504    43     837    2015-08-21 11:48:36 UTC+0000
... 0x86c3a030:taskeng.exe     3384   892    8      89     2015-08-21 11:49:07 UTC+0000
 0x85d01d28:csrss.exe          312    296    9      497    2015-08-21 11:48:35 UTC+0000
 0x86a80920:csrss.exe          412    396    10     301    2015-08-21 11:48:35 UTC+0000
 0x86da6650:winlogon.exe       468    396    6      125    2015-08-21 11:48:35 UTC+0000
 0x86c7c988:GWX.exe            3088   3076   7      205    2015-08-21 11:48:53 UTC+0000
 0x85240d28:explorer.exe       1740   1592   37     874    2015-08-21 11:48:37 UTC+0000
. 0x87066828:vmtoolsd.exe      356    1740   7      179    2015-08-21 11:48:38 UTC+0000
```

psscan – list hidden and terminated processes; for this image does not produce any output.

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

```
Offset(P) Name                PID    pslist psscan thrdproc pspcid csrss session deskthrd
       ExitTime
0x7e29ad28 taskeng.exe        3116   True False True True True True True
0x7e471030 wininit.exe        404    True False True True True True True
0x7e0d39b0 SearchProtocol     2616   True False True True True True True
0x7e08d490 svchost.exe        876    True False True True True True True
0x7e0de518 audiodg.exe        976    True False True True True True True
0x7e588748 SearchFilterHo     2636   True False True True True True True
0x7e3f77d8 svchost.exe        696    True False True True True True True
0x7e5a0c28 GWX.exe            2800   True False True True True True True
0x7e104518 svchost.exe        1004   True False True True True True True
0x7e4cd7d8 svchost.exe        2232   True False True True True True True
0x7e0cf8d8 svchost.exe        1460   True False True True True True True
0x7e064d28 svchost.exe        824    True False True True True True True
0x7e3c56a0 lsass.exe          516    True False True True True True False
0x7e19b030 spoolsv.exe        1284   True False True True True True True
0x7ea72a18 SearchIndexer      2540   True False True True True True True
0x7e0e9030 svchost.exe        1316   True False True True True True True
0x7e3b9d28 services.exe       508    True False True True True True False
0x7e3a4d28 winlogon.exe       468    True False True True True True True
0x7e3d1030 lsm.exe            524    True False True True True True False
0x7e39b308 msdtc.exe          2152   True False True True True True True
0x7e23ed28 ProcessHacker      2920   True False True True True True True
0x7deab320 TPAutoConnSvc      240    True False True True True True True
0x7e4aad28 FTK Imager.exe     3968   True False True True True True True
0x7e100b48 vmtoolsd.exe       1552   True False True True True True True
0x7fc22030 explorer.exe       1764   True False True True True True True
0x7e143030 svchost.exe        1136   True False True True True True True
0x7e051778 svchost.exe        784    True False True True True True True
0x7e5a5030 sppsvc.exe         2832   True False True True True True True
0x7def92a0 vmtoolsd.exe       1412   True False True True True True True
0x7fc07910 dllhost.exe        372    True False True True True True True
0x7ff68030 taskhost.exe       1636   True False True True True True True
0x7e50f7e8 VSSVC.exe          2440   True False True True True True True
0x7e4c66a8 TPAutoConnect      2224   True False True True True True True
0x7ded4d28 WmiPrvSE.exe       252    True False True True True True True
0x7e0107a8 svchost.exe        620    True False True True True True True
0x7fc08030 dwm.exe            1716   True False True True True True True
0x7dede1f8 dllhost.exe        636    True False True True True True True
0x7e4d7d28 conhost.exe        2244   True False True True True True True
0x7e077518 svchost.exe        848    True False True True True True True
0x7e47ad28 csrss.exe          412    True False True True False True True
0x7ef12030 csrss.exe          312    True False True True False True True
0x7f3639e0 smss.exe           228    True False True True False False False
0x7ffc86f0 System             4      True False True True False False False
```

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

```
Offset(P) Name                PID    pslist psscan thrdproc pspcid csrss session deskthrd
       ExitTime
0x7e009100 svchost.exe        624    True False True True True True True
0x7e119600 audiodg.exe        976    True False True True True True True
0x7e149bf0 DumpIt.exe         3568   False False True True True False True
0x7e5df788 SearchFilterHo     2724   True False True True True True True
0x7df08610 dllhost.exe        3312   False False True True True True True
```

```
0x7e0d0518 svchost.exe         832     True False True True True True True
0x7e3a6650 winlogon.exe        468     True False True True True True True
0x7e10aba8 svchost.exe         892     True False True True True True True
0x7de86518 svchost.exe         1124    True False True True True True True
0x7e2e2300 dllhost.exe         2356    False False True True True False True
0x7e174d28 svchost.exe         1164    True False True True True True True
0x7de9c520 dllhost.exe         1584    True False True True True True True
0x7e02a030 svchost.exe         700     True False True True True True True
0x7fc40d28 explorer.exe        1740    True False True True True True True
0x7e090a18 conhost.exe         3576    False False True True True False True
0x7e3c7030 lsm.exe             528     True False True True True True False
0x7e47c6a8 wininit.exe         404     True False True True True True True
0x7e23a030 taskeng.exe         3384    True False True True True True False
0x7e3de6b0 lsass.exe           520     True False True True True True False
0x7e535948 VSSVC.exe           2468    True False True True True True True
0x7e2fc030 dllhost.exe         3032    False False False True True False False
       2015-08-21 11:50:32 UTC+0000
0x7e0c6bb8 svchost.exe         1512    True False True True True True True
0x7e0754a0 TPAutoConnect       2216    True False True True True True False
0x7e27c988 GWX.exe             3088    True False True True True True False
0x7de4c948 TPAutoConnSvc       332     True False True True True True True
0x7df71900 msdtc.exe           2164    True False True True True True True
0x7e0fa650 svchost.exe         868     True False True True True True True
0x7e0cdd28 taskhost.exe        1484    True False True True True True False
0x7e5d8b00 SearchProtocol      2704    True False True True True True False
0x7e03f9e0 svchost.exe         1344    True False True True True True True
0x7e3a1770 services.exe        504     True False True True True True False
0x7e0a9030 spoolsv.exe         1304    True False True True True True True
0x7de66828 vmtoolsd.exe        356     True False True True True True False
0x7e134c70 svchost.exe         1016    True False True True True True True
0x7e062d28 vmtoolsd.exe        1704    True False True True True True True
0x7e5959b0 SearchIndexer       2632    True False True True True True True
0x7dea09f0 dllhost.exe         772     True False True True True True True
0x7fa63a48 sppsvc.exe          3124    True False True True True True True
0x7e1fc478 dwm.exe             1620    True False True True True True False
0x7e07d518 svchost.exe         784     True False True True True True True
0x7f301d28 csrss.exe           312     True False True True False True True
0x7f3d3020 smss.exe            228     True False True True False False False
0x7e480920 csrss.exe           412     True False True True False True True
0x7ffc86f0 System              4       True False True True False False False
0x7de91518                     0       True False False False False True False
```

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:

- pstree 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*.

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

Offset(P)          Name                Base               Size      File
------------------ ------------------- ---------- ---------- ----
0x000000007de011c8  secdrv.SYS          0x97549000          0xa000    \SystemRoot\System32\Drivers\secdrv.SYS
0x000000007de14d68  srvnet.sys          0x97553000          0x21000   \SystemRoot\System32\DRIVERS\srvnet.sys
0x000000007de26468  tcpipreg.sys        0x97574000          0xd000    \SystemRoot\System32\drivers\tcpipreg.sys
0x000000007df0a9c0  srv.sys             0x96235000          0x52000   \SystemRoot\System32\DRIVERS\srv.sys
0x000000007e017130  luafv.sys           0x89895000          0x1b000   \SystemRoot\system32\drivers\luafv.sys
0x000000007e078b48  bowser.sys          0x94d8b000          0x19000   \SystemRoot\system32\DRIVERS\bowser.sys
.
.
.
0x007e2d19c0        DumpIt.sys          0x96316000          0xc0      \??\C:\Windows\system32\Drivers\DumpIt.sys
0x000000007e496f38  cdd.dll            0x94380000          0x1e000   \SystemRoot\System32\cdd.dll
0x000000007e521768  DementiaKM.sys      0x962fa000          0x1c000   \??\C:\d\DementiaKM.sys
0x000000007ea3e008  NDProxy.SYS         0x90f33000          0x11000   \SystemRoot\System32\Drivers\NDProxy.SYS
.
.
.
0x000000007f037e50  usbuhci.sys         0x8ff30000          0xb000    \SystemRoot\system32\DRIVERS\usbuhci.sys
0x000000007f046008  USBPORT.SYS         0x8ff3b000          0x4b000   \SystemRoot\system32\DRIVERS\USBPORT.SYS
```

We scan services using plugin – *svcscan*.

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

Offset: 0x882a40
Order: 413
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: DementiaKM
Display Name: DementiaKM
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_RUNNING
Binary Path: \Driver\DementiaKM
```

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.

```
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.

```
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
 154481104 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:\d\DementiaKM.sys
 503215912 \Device\HarddiskVolume1\d\DementiaKM.sys
```

```
 571430372 \d\DementiaKM.sys
 589959320 \Device\HarddiskVolume1\d\DementiaKM.sys
 598253860 C:\d\DementiaKM.sys
 598254080 (\Device\HarddiskVolume1\d\DementiaKM.sys
 611517522 \??\C:\d\DementiaKM.sys
 741675172 \??\C:\d\DementiaKM.sys
1582231432 \??\C:\d\DementiaKM.sys
1894055248 DementiaKM.sys
1895631246 -d|--driver - name of the kernel-mode driver (default: DementiaKM.sys)
1949748964 \d\DementiaKM.sys
1949749132 \d\DementiaKM.sys
1951321900 \Device\HarddiskVolume1\Windows\System32\catroot\{F750E6C3-38EE-11D1-85E5-
00C04FC295EE}\Package_3_for_KB29\DementiaKM.sys
1951322204 \d\DementiaKM.sys
1960072924 \Device\HarddiskVolume1\d\DementiaKM.sys
1972210148 \Device\HarddiskVolume1\d\DementiaKM.sys
2119309252 DementiaKM.sys
2139533028 \d\DementiaKM.sys
2139533196 \d\DementiaKM.sys
2139533308 \d\DementiaKM.sys
2139533404 \d\DementiaKM.sys
2139776732 \Device\HarddiskVolume1\d\DementiaKM.sys
2143715480 \Device\HarddiskVolume1\d\DementiaKM.sys
2143785252 C:\d\DementiaKM.sys
2143785472 (\Device\HarddiskVolume1\d\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-googlecode\Release\Dementia.pdb
 261026280 d:\fsec2012\dementia\dement~1\objchk_wnet_x86\i386\DementiaKM.pdb
 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’)
2078548456 d:\fsec2012\dementia\dement~1\objchk_wnet_x86\i386\DementiaKM.pdb
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 DementiaKM.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.
 172127474 Dementia.exe
 172128618 DementiaFS.sys
 172129642 DementiaKM.sys
 175243488 DementiaKM
 181843364 Administrator: Command Prompt - Dementia.exe -m 2 -a ‘-p 3324 -p 3856 -P
Dementia.exe -P conhost.exe’
 183391664 C:\d\Dementia.exe
 183391700 \??\C:\d\Dementia.exe
 183391912 Dementia.exe
 183392008 DementiaFS.sys`
 183392104 DementiaKM.sys`
 189969988 m 2 -a ‘-p 3324 -p 3856 -P Dementia.exe -P conhost.exe’
 194812090 DementiaFS.sys
 199059008 Dementia.exe -m 2 -a ‘-p 3324 -p 3856 -P Dementia.exe -P conhost.exe’
 211049176 \Device\DementiaKM
 214121986 Dementia.exe
 214122098 DementiaFS.sys
 214122210 DementiaKM.sys
```

```
 216371130 DementiaKM.sys
 229842704 \Device\HarddiskVolume1\d\DementiaFS.sys
 237026338 \Dementia-1.0-x64.z
 237129728 56 -P Dementia.exe -P conhost.exe’
 240738168 Administrator: Command Prompt - Dementia.exe -m 2 -a ‘-p 3324 -p 38
 243015992 DementiaKM
 243016016 DementiaKM.sys
 243019150 -d|--driver - name of the kernel-mode driver (default: DementiaKM.sys)
 249416116 \DementiaPort
 250469080 Dementia - v1.0 -- Windows memory anti-forensics suite
 250469818 Use: Dementia.exe [-h|--help -d|--debug -f|--file] -m <evasion_method> [-a|--
args]
 261026092 \DosDevices\DementiaKM
 261026140 \Device\DementiaKM
 262873248 Dementia.exe
 265272504 C:\d\Dementia.exe
 266174984 C:\d\Dementia.exe
 419579602 DementiaKM
 419579624 DementiaKM
 430321208 C:\d\DementiaKM.sys
 450568944 \??\C:\d\Dementia.exe
 456898420 DementiaKM
 476186698 Dementia.exe
 481731200 Administrator: Command Prompt - Dementia.exe -m 2 -a ‘-p 3324 -p 3856 -P
Dementia.exe -P conhost.exe’
 484882744 Administrator: Command Prompt - Dementia.exe -m 2 -a ‘-p 3324 -p 3856 -P
Dementia.exe -P conhost.exe’
 486757018 DementiaKM
 486757044 \Driver\DementiaKM
 503022734 Dementia
 503024438 Dementia
 503215912 \Device\HarddiskVolume1\d\DementiaKM.sys
 505081472 \Driver\DementiaKM
 508780838 Dementia
 518175504 DementiaKM
 560621688 \Device\HarddiskVolume1\d\Dementia.exe
 571430372 \d\DementiaKM.sys
 589959320 \Device\HarddiskVolume1\d\DementiaKM.sys
 598253838 DementiaKM
 598253860 C:\d\DementiaKM.sys
 598254080 (\Device\HarddiskVolume1\d\DementiaKM.sys
 599409580 C:\d\Dementia.exe
 604618520 C:\d\Dementia.exe
 611060940 DementiaKM
 611517492 DementiaKM
 611517522 \??\C:\d\DementiaKM.sys
 613400642 Registry\Machine\System\CurrentControlSet\Services\DementiaKM
 614654488 DementiaKMx
 615914188 DementiaKM
 627067444 DementiaKM
 628702404 DementiaKM
 683486000 DementiaKM
 740126244 DementiaKM
 741675172 \??\C:\d\DementiaKM.sys
 744982412 DementiaKM
 746249158 \??\C:\d\Dementia.exe
 746281926 \??\C:\d\Dementia.exe
 747055020 DementiaKM
 747794720 \??\C:\d\Dementia.exe
1582231432 \??\C:\d\DementiaKM.sys
1840800404 DementiaKM
1861412494 DementiaKM
1861412516 DementiaKM
1874465428 DementiaKM
1891444440 Dementia - v1.0 -- Windows memory anti-forensics suite
1891445178 Use: Dementia.exe [-h|--help -d|--debug -f|--file] -m <evasion_method> [-a|--
args]
1893537204 \DementiaPort
1894055224 DementiaKM
1894055248 DementiaKM.sys
1894059004 Dementia-1.0-x64
1895631246 -d|--driver - name of the kernel-mode driver (default: DementiaKM.sys)
1898158264 Dementia.exe
1900341524 Dementia-1.0
1949748964 \d\DementiaKM.sys
1949749132 \d\DementiaKM.sys
1951321900        \Device\HarddiskVolume1\Windows\System32\catroot\{F750E6C3-38EE-11D1-85E5-
```

```
00C04FC295EE}\Package_3_for_KB29\DementiaKM.sys
1951322204 \d\DementiaKM.sys
1960072820 \Device\HarddiskVolume1\d\DementiaFS.sys
1960072924 \Device\HarddiskVolume1\d\DementiaKM.sys
1972210044 \Device\HarddiskVolume1\d\DementiaFS.sys
1972210148 \Device\HarddiskVolume1\d\DementiaKM.sys
1979008578 Dementia.exe -m 2 -a '-p 3324 -p 3856 -P Dementia.exe -P conhost.exe'
1986187812 DementiaKM
2078548268 \DosDevices\DementiaKM
2078548316 \Device\DementiaKM
2113121472 DementiaKM
2119309252 DementiaKM.sys
2139533028 \d\DementiaKM.sys
2139533196 \d\DementiaKM.sys
2139533308 \d\DementiaKM.sys
2139533404 \d\DementiaKM.sys
2139776628 \Device\HarddiskVolume1\d\DementiaFS.sys
2139776732 \Device\HarddiskVolume1\d\DementiaKM.sys
2143715480 \Device\HarddiskVolume1\d\DementiaKM.sys
2143785230 DementiaKM
2143785252 C:\d\DementiaKM.sys
2143785472 (\Device\HarddiskVolume1\d\DementiaKM.sys
2143985818 DementiaKMt
2143985920 :\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\DementiaKM
2143986120 :\REGISTRY\MACHINE\SYSTEM\ControlSet001\services\DementiaKM
2143986330 DementiaKM
2143986356 \Driver\DementiaKM
2145216188 '-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 8bec               MOV EBP, ESP
0x962fca45 83ec0c             SUB ESP, 0xc
```

```
0x962fca48 c645ff00          MOV BYTE [EBP-0x1], 0x0
0x962fca4c 837d0800          CMP DWORD [EBP+0x8], 0x0
0x962fca50 742e              JZ 0x962fca80
0x962fca52 8b4508            MOV EAX, [EBP+0x8]
0x962fca55 50                PUSH EAX
0x962fca56 e8                DB 0xe8
0x962fca57 85                DB 0x85
```

We can do the same with NtWriteFile API:

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

Table below shows parameters that can be used to detect the presence of memory imaging tool.

Tool	Handle	IO	Buffer	Length	Offset	Key	Add. Flags	Process	Ext.	Driver	FILE_OBJECT flags
FTK Imager	UM	UM	UM	0x8000	0	NULL	W,SR,SW	FTK Imager.exe	mem	ad_driver.sys	0x40042
MDD	UM	UM	UM	0x1000	0	NULL	W	mdd_1.3.exe	*	mdd.sys	0x40042
Memoryze	UM	UM	UM	0x1000 mostly	0	NULL	W,SR,SW	Memoryze.exe	img	mktools.sys	0x40042
OSForensics	KM	KM	UM	0x1000	KM	NULL	W	osf32.exe	bin	DirectIo32	0x40062
Win32DD	KM	KM	KM	variable (0x1000 – 0x100000)	KM	NULL	R,W,SR,SW	win32dd.exe	*	win32dd.sys	0x4000a
Winen (EnCase)	UM	UM	UM	totally variable	0	NULL	R,W,SR,SW	winen.exe	E01	winen_.sys	0x40062
Winpmem	UM	UM	UM	0x1000	0	NULL	W,SR	winpmem_...*	*	*(temporary file - random)	0x40042

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
    - Sefault 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:

- Through Control Panel – System and Security – Action Center – Problem Reporting Settings and select 'Never check for solutions'.
- 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

[1] G. Palmer. A Road Map for Digital Forensic Research. Technical Report DTRT0010-01, DFRWS, November 2001. Report from the First Digital Forensic Research Workshop (DFRWS).

[2] Soltan AlharbiJ ens Weber-Jahnke, Issa Traore. 'The Proactive and Reactive Digital Forensics Investigation', 2015. [Online]. http://www.researchgate.net/publication/220849931_The_Proactive_and_Reactive_Dig

[3] Attention Deficit Disorder. Google Code. [Online] https://code.google.com/p/attention-deficit-disorder/

[4] 'Changes to the file system and to the storage stack to restrict direct disk access and direct volume access in Windows Vista and in Windows Server 2008'. [Online]. https://support.microsoft.com/en-gb/kb/942448

[5] David Cowen, Hacking Exposed – Computer Forensic Blog. [Online]. http://www.hecfblog.com/2013/10/daily-blog-130-detecting-fraud-sunday.html

[6] Jason Hale, Digital Forensic Stream. 'The NTFS $LogFile and setMACE'. [Online]. http://dfstream.blogspot.com/2012/01/ntfs-logfile-and-setmace.html

[7] Wicher Minnaard, 'Timestomping NTFS'. [Online] https://www.os3.nl/_media/2013-2014/courses/rp2/p48_presentation.pdf

[8] Ligh, Michael Hale, Andrew Case, Jamie Levy, and Aaron Walters. 'The art of memory forensics: detecting malware and threats in Windows, Linux, and Mac memory'. John Wiley & Sons, 2014.

[9] Wikipedia. [Online] https://en.wikipedia.org/wiki/Billion_laughs

[10] AERAsec Network Services and Security. [Online]. http://www.aerasec.de/security/advisories/decompression-bomb-vulnerability.html

[11] File analysis. Virustotal.com. [Online]. https://www.virustotal.com/en/file/db6981082063dbb4bac89d27c41fbeb86d9e4a97b36661c0945b77a6b9bb0948/analysis/

[12] Pro Hack. Create a Zip Bomb - Zip of Death. [Online]. http://www.theprohack.com/2009/03/create-zip-bomb-zip-of-death.html

[13] Hacking Tutorials. Zip Bomb. [Online]. http://www.theprohack.com/2009/03/create-zip-bomb-zip-of-death.html

[14] Stackoverflow.com. [Online]. http://stackoverflow.com/questions/1459673/how-does-one-make-a-zip-bomb

[15] Wikipedia. [Online]. https://en.wikipedia.org/wiki/Fork_bomb

[16] AERAsec Network Services and Security. [Online]. ftp://ftp.aerasec.de/pub/advisories/decompressionbombs/pictures/

[17] Phil Knufer. 'Mitigating Anti-Forensics: A Schema Based Approach'. [Online]. http://starfish.digitrace.de/media/thesis.pdf

[18] Securityfocus.com. [Online]. http://www.securityfocus.com/bid/3027/exploit/

[19] Didier Stevens, 'Dismantling an XML-Bomb'. [Online]. http://blog.didierstevens.com/2008/09/23/dismantling-an-xml-bomb/

[20] Haking Tutorials. Zip Bomb. [Online]. http://xeushack.com/zip-bomb/

[21] Kat.cr. [Online] https://kat.cr/zip-bomb-insanely-huge-zip-archive-4zb-t2105770.html?download=http://torcache.net/torrent/4CB8D7FCDFAEB5BEB1443AAD59026D7A89D435C5.torrent?title=%5bkat.cr%5dzip.bomb.insanely.huge.zip.archive.4zb

[22] Bugzilla. Mozilla.org. [Online]. https://bugzilla.mozilla.org/show_bug.cgi?id=1031035

[23] Alain Homewood, 'Anti-forensic implications of software bugs in digital forensic tools'. 2012. [Online]. http://aut.researchgateway.ac.nz/bitstream/handle/10292/5364/HomewoodA.pdf?sequence=3

[24] Rapid7.com. [Online]. http://www.metasploit.com/projects/antiforensics

[25] Piper, Davis & Shenoi, 'Countering Hostile Forensic Techniques'. Advances in Digital Forensics II. IFIP international Conference on Digital Forensics. [2006].

[26] Cheong Kai Wee, 'Analysis of Hidden Data in the NTFS File System'. [Online]. http://www.forensicfocus.com/downloads/ntfs-hidden-data-analysis.pdf

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

[28] Wikipedia. [Online]. https://en.wikipedia.org/wiki/Open_source

[29] Scudette in Wonderland. Anti-forensics and memory analysis. [Online]. http://scudette.blogspot.com.ee/2014/02/anti-forensics-and-memory-analysis.html

[30] Handler Diaries. Forensic Analysis of Anti-Forensic Activities. [Online]. http://blog.handlerdiaries.com/?p=363

[31] MalwareJake. Analysis of ADD Ref Image - Part 1. [Online]. http://malwarejake.blogspot.com.ee/2014/01/analysis-of-add-ref-image-part-1.html

[32] Dementia. Google Code. [Online]. https://code.google.com/p/dementia-forensics/

[33] Complete Guide to Anti-Forensics – Leave no trace. [Online]. http://haxf4rall.com/tag/anti-forensics/

[34] Microsoft Developer Network. [Online]. https://msdn.microsoft.com/en-us/library/windows/desktop/aa819132(v=vs.85).aspx

[35] Microsoft Developer Network. [Online]. https://support.microsoft.com/en-us/kb/314834

[36] M. Hammond, 'Python Programming On Win32: Help for Windows Programmers', 2000.