THE REPUBLIC OF TURKEY

BAHCESEHIR UNIVERSITY

INCIDENT RESPONSE - DETECTION AND ANALYSIS ON RECENT VERSIONS OF MICROSOFT WINDOWS

Master's Thesis

EŞREF AYDIN

ISTANBUL, 2018

THE REPUBLIC OF TURKEY

BAHCESEHIR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPUTER ENGINEERING (ENGLISH, THESIS)

INCIDENT RESPONSE - DETECTION AND ANALYSIS ON RECENT VERSIONS OF MICROSOFT WINDOWS

Master's Thesis

EŞREF AYDIN

SUPERVISOR: ASSIST. PROF. AHMET NACİ ÜNAL

İSTANBUL, 2018

THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING (ENGLISH, THESIS)

Name of the thesis: Incident Response - Detection and Analysis on Recent Versions of Microsoft Windows
Name/Last Name of the Student: Eşref AYDIN
Date of the Defense of Thesis: 28.05.2018

The thesis has been approved by the Graduate School of Natural and Applied Sciences

Assist. Prof. Yücel Batu SALMAN
Graduate School Director
Signature

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Arts.

Assist. Prof. Tarkan AYDIN
Program Coordinator
Signature

This is to certify that we have read this thesis and we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Arts.

Examining Comittee Members	Signature
Thesis Supervisor Assist. Prof. Ahmet Naci ÜNAL	----------
Member Assist. Prof. Betul ERDOGDU	----------
Member Assoc. Prof. Bilgin METIN	----------

ABSTRACT

INCIDENT RESPONSE - DETECTION AND ANALYSIS ON RECENT VERSIONS OF MICROSOFT WINDOWS

Eşref Aydın

Computer Engineering (English, Thesis)

Thesis Supervisor: Assist. Prof. Ahmet Naci Ünal

May 2018, 67 pages

Security incidents pose a real threat on organizations and nations. Responding to security incidents quickly is essential to mitigate and limit security incidents negative impacts. However, cyber-crimes are increasing in scale and sophistication and cyber criminals often use different techniques to conceal and remove their traces which make it harder for organizations to detect security incidents and thus to respond to them. Cyber criminals often need to execute malicious programs on targeted systems in different stages of cyber-attack. Thus, extracting information about executed programs on a system is a key element in detection stage of incident response process. With the current fast-paced updates of Windows systems, many incident respond's tools are failing to catch-up with those updates and thus failing to give answers to incident responders that aid them in detection stage of incident response process.

In this research, six different program execution artifacts, on latest version of Windows at the time of this writing Windows 10 1079, were analyzed in depth. For each artifact, the storage location was identified, the stored data format and structure was studied, and often reverse-engineering process was required. In addition to that, each artifact was compared against ten different items of forensics interests. Then, results of running well-known incident parsing tools on those artifacts were evaluated. As a result of this research, it was found that parsing tools often fail to extract all possible information stored in program execution traces. Also, the conclusion of some researches, that were done on extracting program execution traces, were found to be inaccurate and providing misleading information.

Keywords: Incident Response, Windows execution artifacts

ÖZET

OLAY MÜDAHALE - MİCROSOFT WİNDOWS' UN SON SÜRÜMÜ ÜZERİNDE TESPİT VE ANALİZ

Eşref Aydın

Bilgisayar Mühendisliği (İngilizce Tezli)

Tez Danışmanı: Dr. Öğr. Üyesi Ahmet Naci Ünal

Mayıs 2018, 67 sayfa

Güvenlik olayları organizasyonlar ve ülkeler için büyük bir tehdit oluşturmaktadır. Güvenlik olaylarına hızlı bir şekilde yanıt vermek, olayların olumsuz etkilerinin azaltılması ve sınırlandırılması için çok önemlidir. Bununla birlikte siber suçlar belirli ölçekte ve karmaşıklıkta artmaya devam etmektedir. Siber suçlular izlerini gizlemek veya yok etmek için farklı teknikler kullanarak organizasyonların olayları tespit etmesini ve dolayısıyla çözümlemesini zorlaştırmaktadır. Siber suçluların genellikle siber saldırıların farklı aşamalarında hedeflenen sistemler üzerinde kötü amaçlı programlar çalıştırması gerekir. Bu sebeple, bir sistem üzerinde çalıştırılan programlar hakkında bilgi elde edebilmek, olay müdahale sürecinin tespit aşamasında önemli bir kilit noktasıdır. Günümüzde Windows sistemlerinin mevcut hızlı güncellemeleri ile, birçok olay müdahale aracı bu güncellemeleri yakalayamamakta ve bu nedenle de olay müdahale sürecinin tespit aşamasında müdahale ekibine gerekli bilgiyi sağlayamamaktadır.

Bu araştırmada, incelemenin yapıldığı sırada en son Windows sürümü olan Windows 10 1079 için sistem üzerinde altı farklı uygulama çalıştırma artifact' ı derinlemesine incelenmiştir. Her artifact için veri depolama lokasyonu belirlenmiş, depolanmış veri formatı ve yapısı incelenmiştir ve bu süreçte çoğunlukla tersine mühendislik yöntemi kullanılmıştır. Buna ek olarak her artifact adli değere sahip 10 farklı bilgi için karşılaştırılmıştır. Daha sonra, bu artifact' lar iyi bilinen veri ayrıştırma araçları ile çalıştırılarak sonuçları değerlendirilmiştir. Bu araştırmanın sonucunda, çoğu kez adli bilişim ayrıştırma araçlarının uygulama çalıştırma izlerinde saklanan tüm olası bilgileri elde edemediği bulunmuştur. Ayrıca, uygulama çalıştırma izlerinin çıkarılması sırasında yapılan incelemelerde bazı sonuçların yanlış olduğu ve yanıltıcı bilgi sağlandığı tespit edilmiştir.

Anahtar Kelimeler: Olay Müdahale, Windows uygulama çalıştırma artifact' ları

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

APT	:	Advanced Persistent Threat
FAT	:	File Allocation Table
FTK	:	Forensic Toolkit
GB	:	Gigabyte
HKCU	:	HKEY Current User
HKLM	:	HKEY Local Machine
IOC	:	Indicator of Compromise
ISO	:	International Organization for Standardization
MS-DOS	:	Microsoft Desktop Operating System
MSDN	:	Microsoft Developer Network
NIST	:	National Institute of Standards and Technology
NTFS	:	New Technology File System
P2P	:	Peer to Peer
PCA	:	Program Compatibility Assistant
SCADA	:	Supervisory Control and Data Acquisition
UTF	:	Unicode Transformation Format
Windows NT	:	Windows New Technology

1. INTRODUCTION

Nowadays, with the presence of Advanced Persistent Threat (APT) groups, cyber-attacks are getting more sophisticated and increasingly harder to detect by targeted parties. APT is defined as "deliberately slow-moving cyberattack that is applied to quietly compromise interconnected information systems without revealing itself" (Friedberg, et al., 2015). APT groups are often sponsored by governments or other large players and thus, they have virtually unlimited resources and advanced technological knowledge. During different stages of cyber-attack, APT groups often need to execute malicious tools on targeted systems and they need to do so in stealthy manner to avoid detection and keep their foothold in targeted organization network. To do so, APT groups often use different anti-forensics technique to conceal program execution traces which makes it harder for targeted organization to discover the breaches. Studies show that organizations take several months to detect security incident. For example, "Ponemon Cost of Data Breach" study states that organizations take 206 days on average to detect a breach (IBM, 2017).

Security incidents often have devastating impacts on targeted parties. Negative impacts of security incidents may include loss of reputation, safety issues, legal penalties, direct financial loss and business productivity (Alberts & Dorofee, 2002). In addition to that, security incidents pose a real threat to national security like the case of Stuxnet which targeted nuclear facilities in Iran. Recent security incidents which targeted high-profile organizations, affected millions of people worldwide and resulted in direct financial losses of millions of dollars. One example is Yahoo data breach, which was announced in 2016, costed Yahoo 350$ million and compromised over 1 billion Yahoo accounts information (Newman, 2016). So, to minimize the negative impacts of security incidents, organizations must detect and act on those incidents quickly. Since executing malicious tools is part of almost every security incident, checking and finding executed programs on systems is critical part of detecting security incidents.

Since Microsoft Windows is the dominate desktop operating system, with holding over 82% of market share (statcounter.com, 2018), this research will focus on detecting executed

programs on Microsoft Windows. For different purposes like improving performance and user experience, Windows keeps track about executed programs by save information about those programs in different locations on system. For example, to allow users to launch their recently used programs quickly, Windows keeps track of recent programs in Registry. The data stored about executed programs depends Windows's feature or technology that saved this data. For example, while recent programs list saves program last execution date, performance related features often save a list DLLs that were loaded by the executed program. For this reason, all sources of information about executed programs need to be examined carefully and correlated together to give incident responders full picture of programs that were executed during incident and increase their confidence with their findings.

1.1 AIMS

With current fast-paced updates to Windows operating system, the number of program execution's information sources and the format of saved data are often getting changed with newer Windows versions. Thus, many available parsing tools are failing to catch-up with those updates and consequently failing to provide incident responders with full information available about executed programs in Windows system. In addition to that, the exact meaning of saved data, about executed programs, is sometimes misunderstood and poorly documented which leave incident responder with limited options when available tools fail to extract executed programs information on recent Windows versions.

This research will examine saved data about executed programs in all available sources of information on latest version of Microsoft Windows at the time of this writing, Windows 10 1097. The binary data structures will be reverse-engineered to get human readable information and exact meaning of data will be documented. Also, this research defines ten items of forensics importance about executed programs and will evaluate each program execution artifact against those items. In addition to that, several available parsing tools are evaluated against each studied program execution.

The aim of this research is to provide incident responders will full and accurate information about executed programs saved data on latest version of Windows. And thus, reducing needed time to detect potential security incidents on Window systems.

1.2 THESIS STRUCTURE

The next chapter are organized as the following:

a. Chapter 2 - Literature Review: Start with exploring security incident definitions and its negative impact on organizations. This chapter also lists examples about high-profile security breaches that happened in recent years. In addition to that, the important of incident response model is shown along with exploring different stages of incident response process. After that, Microsoft Windows history and basic terminology are explained.
b. Chapter 3 - Data and Method: Applied testing mythology's stages are discussed. Test and analysis machines setup and used tools are listed. Results verification and limitations of study were mentioned.
c. Chapter 4 - Findings: In this chapter, six different program execution artifacts are analyzed, and their saved data structure is explored. Those artifacts are checked against ten different forensics item of interests that are introduced in that chapter. Also available parsing tools' ability to parse those artifacts is evaluated.
d. Chapter 5 - Conclusion and Discussion: This chapter provide a summary of results obtained during this research.

2. LITERATURE REVIEW

In this chapter, security incident definition from academia and industry will be explored. In addition to that, security incident negative impacts on organization are discussed. Then, a general incident response process model and the definition of stages and actions involved in each stage are introduced. The last section of this chapter will be for Microsoft Windows, its market share, versions and important features.

2.1 INFORMATION SECURITY INCIDENT

2.1.1 Definition

There are several definitions of information security incidents. For example, (Cichonski, et al., 2012) define computer security incident as "a violation or imminent threat of violation of computer security policies, acceptable use policies, or standard security practices". Whereas, (ISO, 2018) sees information security incident as "single or a series of unwanted or unexpected information security events that have a significant probability of compromising business operations and threatening information security". (ISO, 2018) goes ahead and defines security event as: "identified occurrence of a system, service or network state indicating a possible breach of information security policy or failure of controls, or a previously unknown situation that can be security relevant". Another definition for security incident is from (Ahmad, et al., 2012) as "An information security incident occurs when there is a direct or indirect attack on the confidentiality, integrity and availability of an information asset". Security incident definition may also change from country to country and from one field to another. So, having several definitions of security incident, requiring from organizations to have precise and clear definition of security incidents in order respond to security incidents efficiently and deals with their legal consequences. Some examples of computer security incidents include:

a. Infecting a computer system with malware
b. Using computer system to attack other computer systems without owner knowledge.

c. Violate corporate policies by using corporate computer in illegal activities like pirated content P2P, surfing adult websites...etc.
d. Ransom based attacks.
e. Stealing sensitive proprietary information.

There are many other examples of information security incidents. However, in this thesis only information security incidents that involve launching executable modules on target computer will be researched.

2.1.2 Security Incidents Impact

Information security incidents can have devastating consequences on organizations. (Alberts & Dorofee, 2002) define several areas of organizations where information incident security incidents can have negative impacts:

a. Reputation and customer confidence
b. Safety and health issues
c. Fines and legal penalties
d. Financial
e. Productivity

In recent years many large-scale information security incidents targeted big organizations and had major effects on those organizations. For example, in 2016 internet giant Yahoo disclosed that over 1 Billion user accounts were compromised (Newman, 2016). In addition to huge impact on Yahoo reputation, Yahoo has direct financial loss of over $350 million due to this security incident (Lunden, 2017). Similar information security incident is Sony PlayStation network breach which happened in April 2011 (Thomas, 2011). 77 million accounts were hacked, and website went down for months. This security incident loss was estimated to be around $171 million (Armerding, 2018). Another example that shows other impacts of security incidents is Stuxnet. Stuxnet was first discovered in 2010 by Kaspersky expert Sergey Ulasen (Kasperksy, 2011). Stuxnet is considered as the world first digital weapon and it targeted SCADA systems at Iranian nuclear facilities, resulting in failure of those systems which in turn had negative impacts on nuclear facilities operations. Healthcare sector was also appealing target for cyber criminals. In 2015, Anthem

announced that it was victim of major security incident. Around 78 million of customers records were exposed (Ragan, 2015). These records included information such as cumber name, social security number, date of birth and possible medical history. This breach costed Anthem around $100 million in addition to negative impact on its reputation.

2.1.3 Importance of Security Incidents Response

Information security incidents examples discussed in previous section show the huge impacts that those security incidents can have on organizations. Thus, it’s important for organizations to be prepared for such incidents and have well trained incidents response teams and ready plans in hands to respond to security incidents probably and as fast as possible to mitigate their negative impact on organization.

In addition to mitigating negative impact, security incidents response plans are now mandatory for some sectors in some countries. For example, section 164.308 (a)(6)(ii) of security rules for "Health Insurance Portability and Accountability Act" (HIPAA), from (Code of Federal Regulations (CRF), 2013), states that:

"Implementation specification: Response and reporting (Required). Identify and respond to suspected or known security incidents; mitigate, to the extent practicable, harmful effects of security incidents that are known to the covered entity or business associate; and document security incidents and their outcomes."

2.1.4 Security Incident Response Process

Many incident response process models were introduced. Some examples are: ISO/IEC 27035-1:2016: Information security incident management -- Part 1: Principles of incident management (ISO, 2016), NIST SP 800-61 - Computer Security Incident Handling Guide (Cichonski, et al., 2012) and SANS Incident Handling Process (SANS Institute, 2017). Those process models have some differences either in number of stages or definitions and tasks involved in those stages (Grispos, 2016). Thus, there one accepted process model that be considered as industry-standard model for incident response (Mitropoulos, et al., 2006). However, majority of available models involve "Detection" stage, and this thesis will only

focus on executable modules execution detection which takes place in "Detection" stage. So, exact differences between available incident response models are not of interest in this research. (Grispos, 2016) compared ten different incident response process models and concluded that typical incident response model is consisted of six stages:

2.1.4.1 Preparation

In this stage, security hardware and software are installed configured on organization network. These security controls include firewalls, intrusion detection systems, monitoring tools and they must be configured in a way that provide security team with required information in case of security incident. In addition to that, the security incident response team is created, and its responsibility is defined precisely. Security incident response team should get required support from management to get required privileges and permissions to be able to operate quickly and provided with required tools to respond to security incidents when they happen.

2.1.4.2 Detection

This stage happens when security incident response team becomes aware of security incident. This can be done in several ways, either through reporting by internal employee or external entity, such as law enforcement, or through detection by incident response security team. Incident response security team can detect security incidents by searching for IOC found in security controls systems' logs, that were installed in "Preparation" stage, or on end-points systems. Digital forensics will often need to be used in this stage to detect possible security incidents. This research will focus on finding executed programs traces at Microsoft Windows endpoint systems.

2.1.4.3 Containment

In this stage, security incident response team works to limit and control negative impact of security incidents. Actions taken in this stage, include things like isolating affected systems or subnets, disabling user account, changing passwords...etc.

2.1.4.4 Eradication

Next step after containing and control security incident is to eliminate it completely. In this stage, security incident response team takes actions such as removing malware, reinstalling systems, applying patches, disabling affected services and so on (Cichonski, et al., 2012) .

2.1.4.5 Recovery

Actions taken in this stage overlap with actions taken in "Eradication" stage. While Crispos sees "Eradication" and "Recovery" (Grispos, 2016) to be separated stages, NIST combines them in one stage "Eradication and Recovery" (Cichonski, et al., 2012). The main objective of this stage is to recover normal working conditions to systems affected by security incidents. Actions taken include rebuilding systems, recovering data from backups, reconnect systems to network and so on.

2.1.4.6 Follow-up

This stage is also referred to as lessons learned stage. In this stage, security team analyzes and documents how security incident happened and what actions need to be taken in order to eliminate future threats of similar incidents. Other actions that are taken in this stage include finding new IOCs and feed security control systems for those systems to detect and stop future attacks.

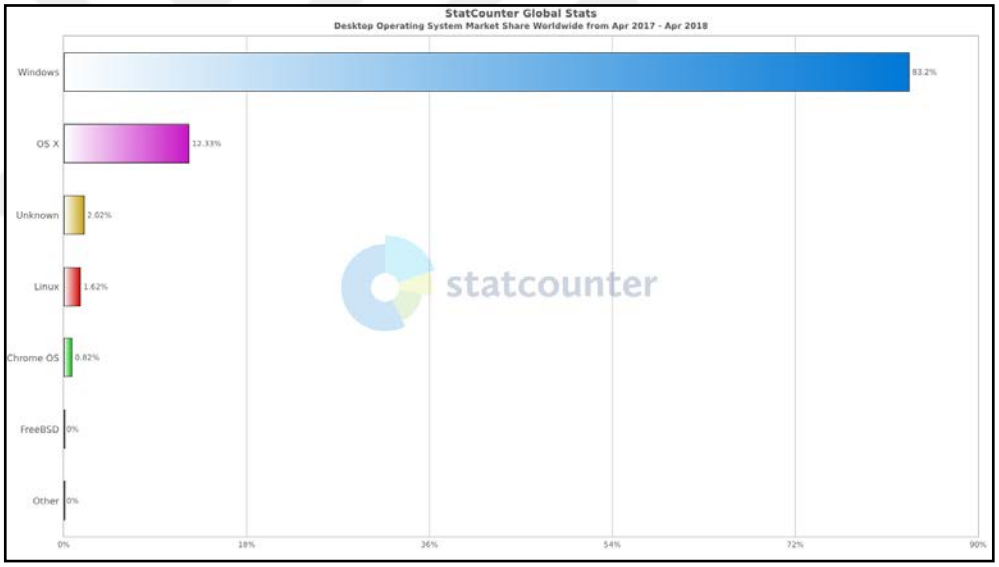
2.2 MICROSOFT WINDOWS

Microsoft Windows was first released in 1985 as graphical interface for MS-DOS operating system (Calore, 2008). Later in 1995, with the release of Windows 95, Microsoft Windows was developed to be fully functional operating system on its own rather than being merely graphical interface extension of MS-DOS like previous versions of Windows. In addition to Windows 9x series which was focused on average consumer, Microsoft introduced Windows NT series which was designed to run on high-end workstations in organizations environment (Calore, 2008). Windows NT series was introduced with the release of Windows NT 3.11, which included advanced networking capabilities and introduced new

file system: NTFS to replace FAT file system used on MS-DOS and Windows 9x series. Microsoft kept maintaining two series of Windows, the Windows 9x series and Windows NT series, till the release of Windows XP in 2001. Microsoft Windows XP was built on Windows NT Kernel and included consumer-oriented features from Windows 9x series.

Since the release of Microsoft Windows, it dominated the market of desktop operating systems. Past year's usage statistics shows that Microsoft Windows has over 82% of global market share of desktop operating systems followed by macOS X which hold 12.33% of market share (statcounter.com, 2018) as Figure 2.1 shows.

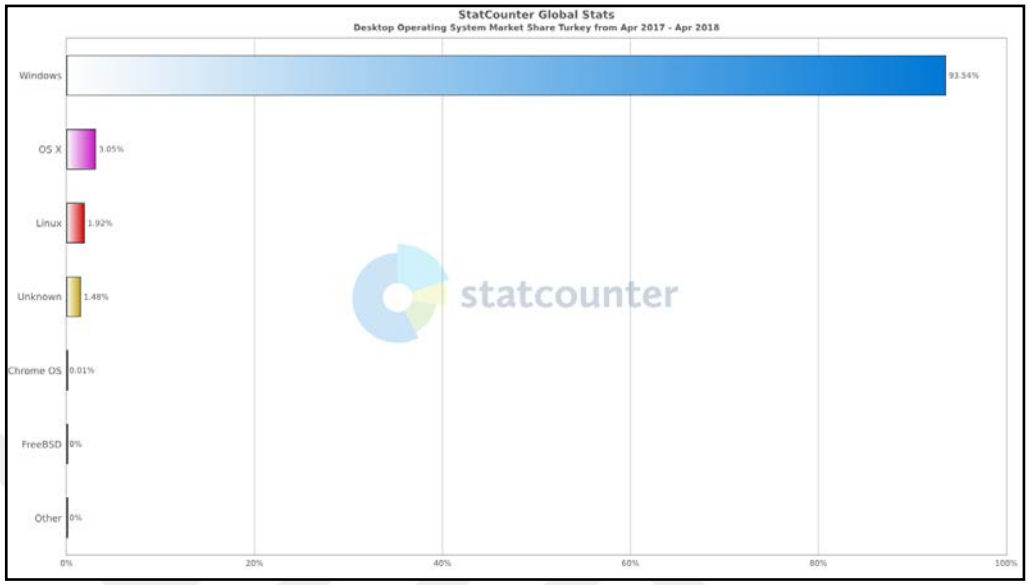
Figure 2.1: Desktop operating systems market share worldwide



Source: statcounter.com

In turkey, desktop operating systems usage statistics show even higher market share for Microsoft Windows of over 93% (statcounter.com, 2018) as Figure 2.2 shows.

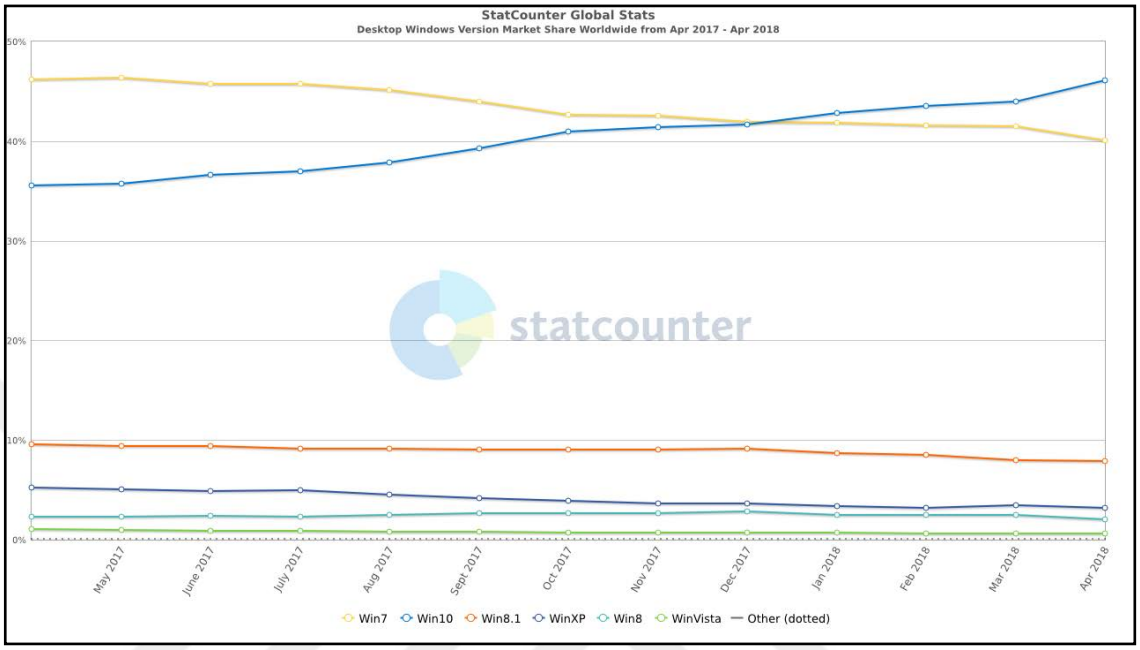
Figure 2.2: Desktop operating systems market share turkey



Source: statcounter.com

The last version of Windows at the time of this writing is Windows 10 version 1097. Windows 10 introduced several new features and significant security enhancement to Microsoft Windows 8.1. These features include new internet browser Edge to replace Internet Explorer 11, personal assistant Coratan and Universal Apps. In addition to that, Microsoft Windows 10 is based on OneCore kernel which means that Microsoft is maintaining one version of Windows to run on several different platforms, including Xbox gaming station, desktop, mobile and IoT devices. Figure 2.3 shows that In December 2017, Windows 10 overtook Windows 7 in market share to become the most used Windows version. Current usage share of Windows 10 is over 46% (statcounter.com, 2018)

Figure 2.3: Desktop windows versions market share worldwide



Source: statcounter.com

Since Windows 10 is future of Windows and is rapidly becoming the most used Windows version, this research will focus only on analyzing execution traces on Windows 10.

2.2.1 Windows Registry

Windows registry is a core component of Windows systems (Carvey, 2016) and it's used to store Windows and programs configurations and it's stored in binary files called hives. Windows registry have hierarchical structure consisting of keys and values, where each key can hold sub keys, values or both. Registry keys have a last update timestamp that stores last time the key was changed. Registry values are used to hold different types of database, like binary, plain text or numbers. Windows registry is of outmost importance in digital forensics and incident response as it hold a wealthy of data related to Windows system activity.

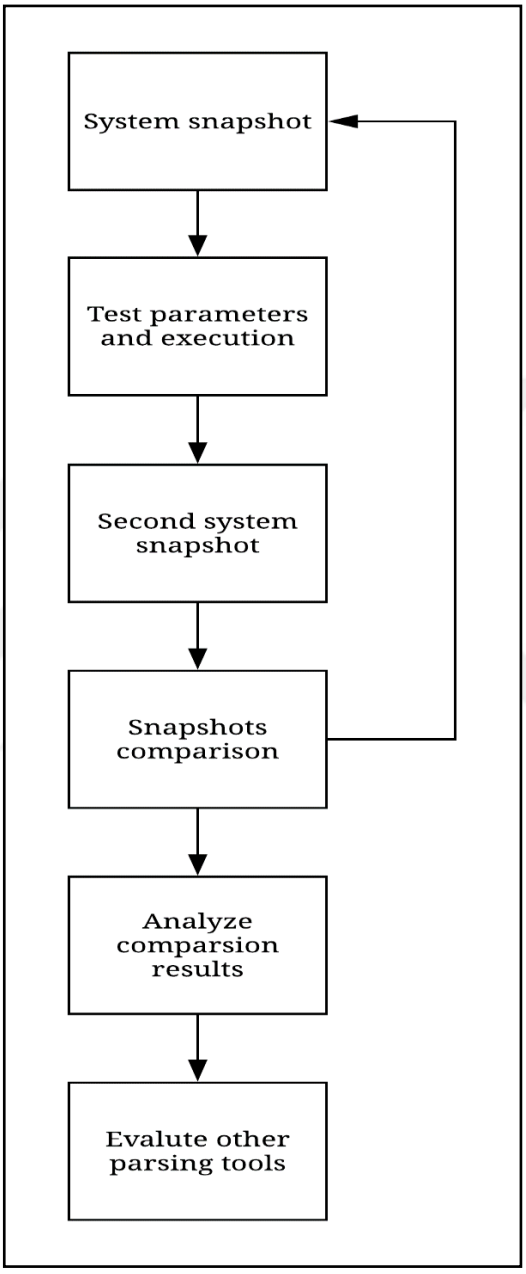
3. DATA AND METHOD

This research will deal with programs execution artifact from incident responders point of view. Thus, data acquisition methods and tools will be used accordingly without taking into consideration typical digital forensics principles in data acquisition.

The general methodology has five stages:

i. Take snapshot of current Windows system status
ii. Execute artifact specific scenario and note test parameter, like execution time, executable name and so on.
iii. Take another snapshot of Windows system status
iv. Compare snapshots taken from step#i and step#iii
v. Analyze differences found in step#iv and extract information from them.
vi. Run other parsing tools and compare results

Figure 3.1: Analysis process



Stages shown in Figure 3.1 are discussed again in "Findings" chapter. For some artifacts those stages need to be tweaked and additional stages might be necessary.

In the remaining part of this chapter, different tools used in this research are introduced and more details are provided on some stages.

3.1 SOFTWARE AND ENVIRONMENT SETUP

During this research, two Windows systems were used. One system used to create artifacts data by executing programs with different scenario, the other system is used analyze artifacts data. On each system, different tools are used to complete this research.

3.1.1 Test Machine Environment Setup

Test machine is a virtual machine created with VMware Workstation and it has the following specifications and tools:

a) Windows 10 Professional 64bit, version 1097.
b) 8 GB of physical memory.
c) Sysinternals "Process Monitor v3.50": Which is used to monitor running process's file system, registry and processes and other general activities.
d) Syinternals "Process Explorer v16.21": Considered as the most advanced Windows process management tool. "Process Explorer" provide details about running processes, such CPU and memory usages, running threads inside process, process's opened handles, loaded modules…etc.
e) "Regshot v1.9": Regshot is open source tool used to take snapshot of Windows registry and to compare different snapshots and list the differences.
f) "Regedit": Windows integrated registry editor program.
g) "reg save": Windows integrated tool and is used in this research to save snapshot of Windows registry in hive format along with "Regedit" tool.

3.1.2 Analysis Machine Environment Setup

The machine that used to analyze extracted from test machine has the following specification and tools:

a) Windows 10 Home 64bit, version 1097.
b) 16 GB of physical memory
c) "WinMerg v2.14": Used to compare text based files.

d) "010 Editor v8.0.1": Hex editor used to do low level analyses and reverse-engineering of binary data.
e) "VMware Workstation Pro v14.1.1": Used to run test machine.
f) "FTK Imager v3.1.4.6": Used to extract files from test machine virtual disk.
g) "Forensics Explorer v1.0.0.2": Powerful offline registry viewer tools with many forensics related features. It also uses plugins to parse several artifacts in registry and show them in human readable form.
h) "RegRipper v2.8": Open source registry forensics framework that uses plugins to parse artifacts from registry.
i) "AXIOM v1.2.2": Well-known digital forensics software which supports hundreds of artifacts and is used by thousands of customers.
j) "TZWorks": Group of separate tools used parse artifacts, where each tools is specialized with parsing one artifact.

3.2 TEST SCENARIOS

For each artifact multiple scenarios were performed. Following are some of those scenarios:

a) Execute program with normal condition
b) Move program to different location on file system and execute it.
c) Execute same program multiple times.
d) Change program file name and execute it.
e) Change program file's timestamps and execute it.
f) Use different program and rename it with same file name as tested program and execute it.
g) Do above scenarios once without rebooting the machine again after rebooting the machine and note any changes.

More test scenarios are added in "Findings" chapter.

3.3 RESULTS VERIFICATION

To verify analysis findings, the original scenarios' parameters are compared against information extracted from saved data. For example, if particular artifact contains last execution timestamp, the extracted timestamp value is compared against actual last execution time of the studied program that were noted during test scenario.

3.4 RESEARCH LIMITATIONS

In this research, only Windows 10 version 1097 is considered. This is because, Windows 10 is gaining more popularity over other Windows versions and to limit the scope of this study. Also, only two commercial forensics tools were considered: AXIOM and TZWorks. Other well-known digital forensics tools were not checked because it wasn't possible to get trial versions.

4. FINDINGS

In this chapter, program execution traces on Microsoft Windows are studied and analyzed. Six different artifacts are checked. Each artifact has a separate section, in which each artifact's storage location and stored data structure is analyzed and reverse-engineered if necessary. Stored artifact's data is examined to find out what forensics items of interests are stored. Ten different forensics items of interests, about executed programs, are studied. Those items are:

a. Executable file name: The executed program's file name.
b. Executable file path: This piece of information is necessary as attackers often name their malicious tools with well-known legitimate executables but run them from different location on file system.
c. Executable file size: File size of executed program can give indication
d. Executable file date: This can be any of file MACB timestamps stored in file system. For example, NTFS file system stores MACB for file in two different locations: $STANDARD_INFORMATION and $FILE_NAME. MACB timestamps are:
    i. Modified: file's data last modification date.
    ii. Accessed: file last accessed date. In recent versions of Windows this value is not updated.
    iii. Changed: MFT entry last changed date.
    iv. Birth: file creation date.
e. Executable file hash
f. Execution date: Last execution date of the program.
g. Execution history: Historical execution records of the program.
h. Executions count: Number of times the program has been executed.
i. User: User account who executed the program.
j. Requires reboot? Whether this artifact requires reboot to save its data.

After artifact analysis is done, different well-known forensics tools' ability, to parse this artifact's format on latest version of Windows, is tested.

4.1 PROGRAM COMPATIBILITY ASSISTANT

Microsoft Windows Vista has introduced new features and changes to Windows kernel, aiming especially to enhance security. During Microsoft internal testing of Windows Vista, they found out that some applications, which work on Windows XP, are no longer working Windows Vista. To solve this problem, Microsoft introduced a new technology called Program Computability Assistant or PCA.

Program Compatibility Assistant runs on Windows as a service under "svchost.exe"

When user executes a program, PCA automatically checks for known compatibility issues and tries to solve those issues to enable legacy program to run smoothly on newer versions of Windows operating system. PCA may also block the program completely from running if it has a serious known compatibility issues, which in this case is called hard block. To enhance user experience and performance, PCA saves data about executed programs in several different places on Windows. This data can be used forensically to get valuable information about executed programs on Windows operating system.

A lot of research has been done on PCA artifacts. However, it was found that those researches are not complete that they are either not valid on recent version Windows, don't cover all PCA artifacts or don't go in details of binary structures and thus don't explain all important data related to incident response.

During analysis of two Windows systems, PCA artifacts were identified and listed in Table 4.1

Table 4.1: Program Compatibility Assistant related registry keys

Artifact	Registry key
AppCompatFlags - Store	HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store

AppCompatFlags - Persisted	HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Persisted HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Persisted
AppCompatFlags - Layers	HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers
Shimcache	HKLM\SYSTEM\ControlSet001\Control\Session Manager\AppCompatCache

4.1.1 AppCompatFlags - Store

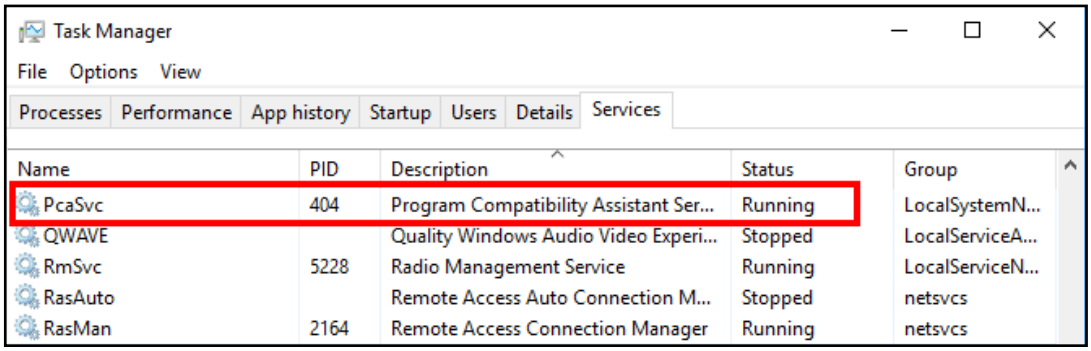
4.1.1.1 Analysis

In recent versions of Windows, PCA was found to store data about executed programs under "HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store" registry key (Harrell, 2013). Unlike many other program execution artifacts, "AppCompatFlags - Store" is saved under "HKCU", which can be used to relate executed program to specific user account.

To understand behavior of PCA and "AppCompatFlags - Store" key, "vmmap.exe" program execution were studied at program start and shutdown using Process Monitor tool.

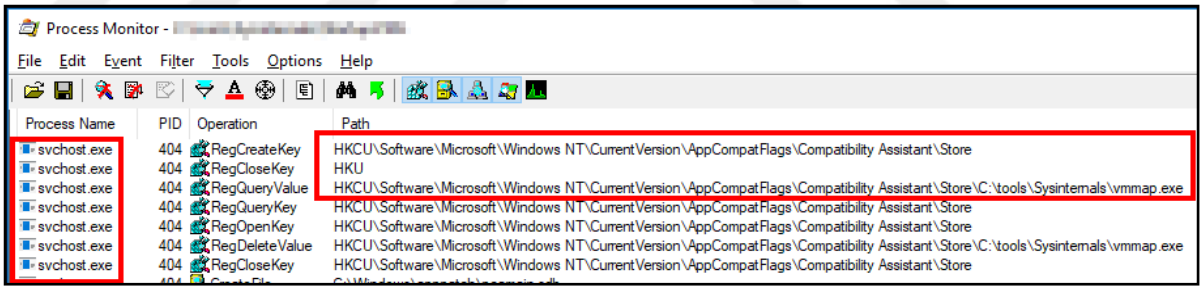
As mentioned earlier, PCA is running as a service under "svchost.exe". However, since there are many "svchost.exe" processes running on Windows system at any given time, another tool, Windows Task Manager, need to be used to find exact "svchost.exe" Process ID.

Figure 4.1: Program Compatibility Assistant service's process ID



After finding "svchost.exe" PID, as shown in Figure 4.1, "vmmap.exe" was launched to monitor PCA behavior on program execution.

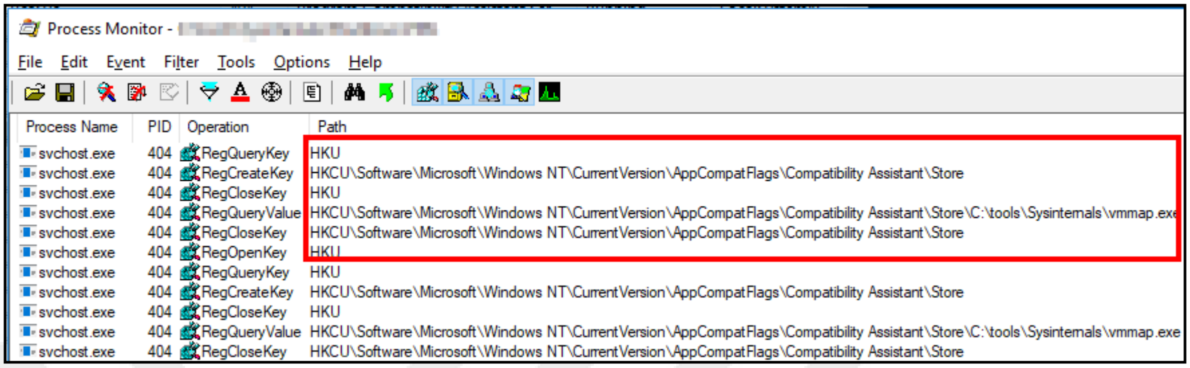
Figure 4.2: Program Compatibility Assistant service queries "Store" key on program execution



As Figure 4.2 shows, PCA query "HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store" key several times upon program execution. It also checks for registry value named "C:\tools\Systinternals\vmmap.exe" under that key. The registry value name is the full path of the executable file. If this value doesn’t exist, PCA creates it, otherwise PCA updates this value.

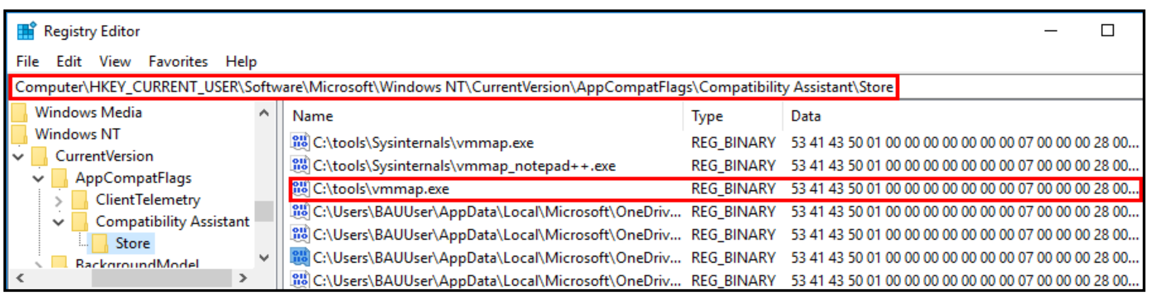
During program shutdown, PCA acts in similar way to "AppCompatFlags\Compatibility Assistant\Store" key as it does during startup.

Figure 4.3: Program Compatibility Assistant service queries "Store" key on program shutdown



PCA, again queries "AppcompatFlags\Compatibility Assistant\Store" key few times and if "C:\tools\Systinternals\vmmap.exe" doesn't exist, PCA creates this value as shown in Figure 4.3.

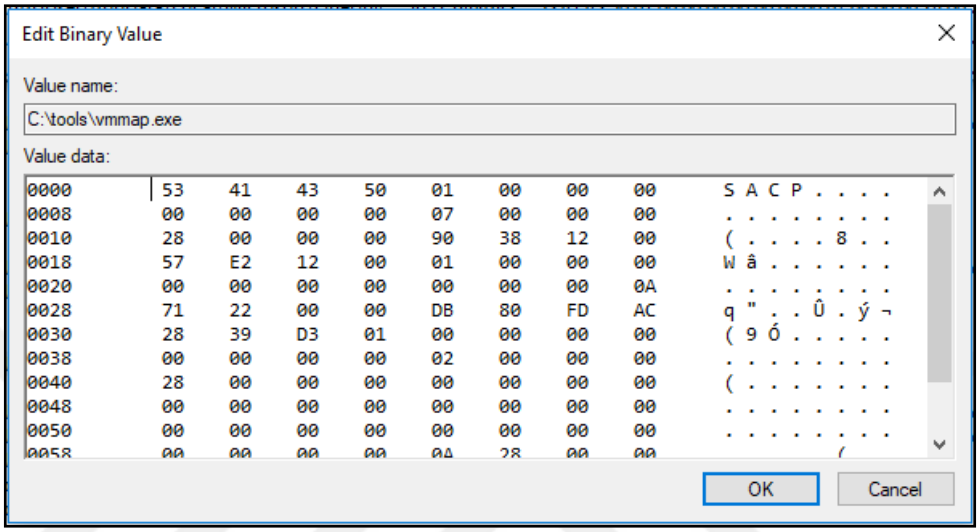
Figure 4.4: "Compatibility Assistant\Store" registry key's contents



"AppcompatFlags\Compatibility Assistant\Store" key contains a list of registry values of type "REG_BINARY" which holds binary data as shown in Figure 4.4. Those values are created first time a program is executed and are updated upon subsequent program executions.

The value name is the full path of executed program, which is valuable information for incident responders. Value data on the other hand, is binary data. Figure 4.5 shows sample value data

Figure 4.5: "Compatibility Assistant\Store" registry key sample value's data



Harrell mentioned that "AppcompatFlags\Compatibility Assistant\Store" key is being used in recent Windows versions instead of "Persisted" key (Harrell, 2013). However, no explanation of binary data was provided. In order to extract human readable information, binary data needs reverse-engineering.

A list of programs was executed clean Windows 10 system and resulted binary values were examined and compared to understand their meaning. Table 4.2 is a list of programs that were executed.

Table 4.2: Executable name and size values

Executable Name	Executable Size (Bytes)
advancedrun.exe	89,296
alternatestreamview.exe	49,360
appcrashview.exe	48,848

awatch.exe	35,328
axhelper.exe	39,424
browseraddonsview.exe	456,912
browsinghistoryview.exe	499,408
bulkfilechanger.exe	135,376
bulletspassview.exe	98,400
Bginfo.exe	3,001,480
vmmap.exe	1,194,128
RAMMap.exe	625,816
procexp64.exe	1,462,272
npp.6.7.4.Installer.exe	7,965,917

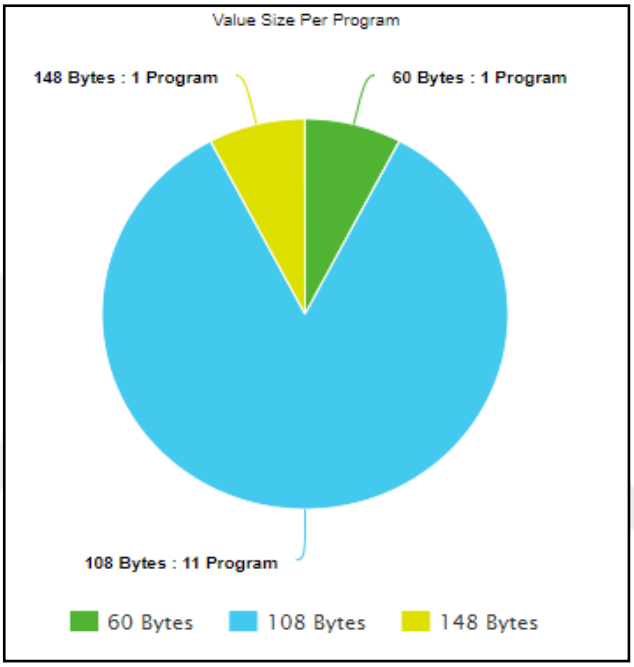
Resulted binary data had variable size. Three sizes were observed and listed in Table 4.3.

Table 4.3: "Compatibility Assistant\Store" values' data size of executables

Data size (Bytes)	Programs
60	npp.6.7.4.Installer.exe,
108	advancedrun.exe, alternatestreamview.exe, awatch.exe, axhelper.exe, browseraddonsview.exe, browsinghistoryview.exe, bulkfilechanger.exe, bulletspassview.exe, Bginfo.exe, vmmap.exe, RAMMap.exe
148	procexp64.exe

Figure 4.6 shows a bie chart of programs counts and their corrponding values size.

Figure 4.6: "Compatibility Assistant\Store" key's values size



Most programs execution result in value size of 108 bytes. Also, it was observed that:

a. There is no relation between program executable size and registry value size.
b. There is no relation between program architecture (32x or 64x) and registry value size.

Value's binary data was exported and examined in external hex-editor in order to facilitate reverse-engineering process.

Figure 4.7 is example of 60 bytes length value:

Figure 4.7: "Compatibility Assistant\Store" registry key - 60 bytes sample

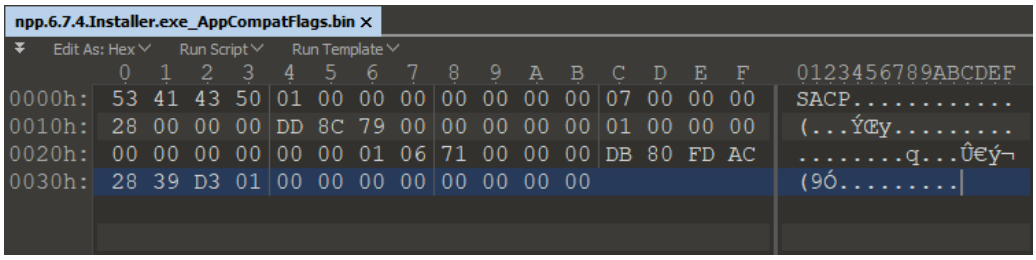


Figure 4.8 is example of 108 bytes length value:

Figure 4.8: "Compatibility Assistant\Store" registry key - 108 bytes sample

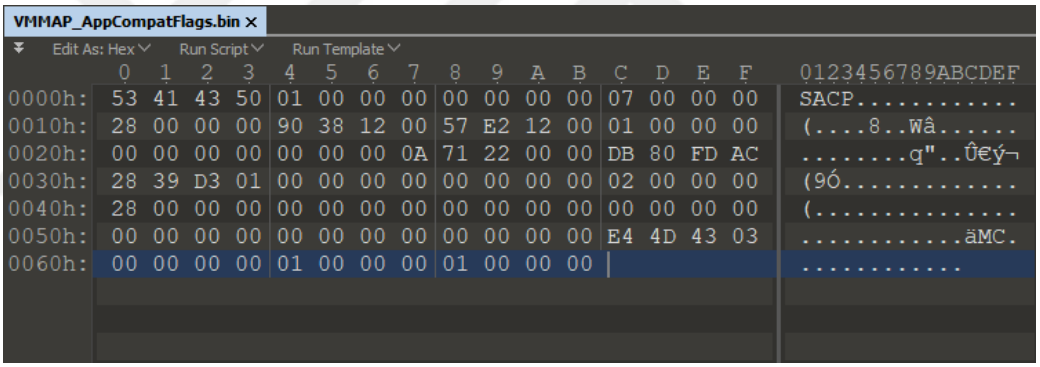
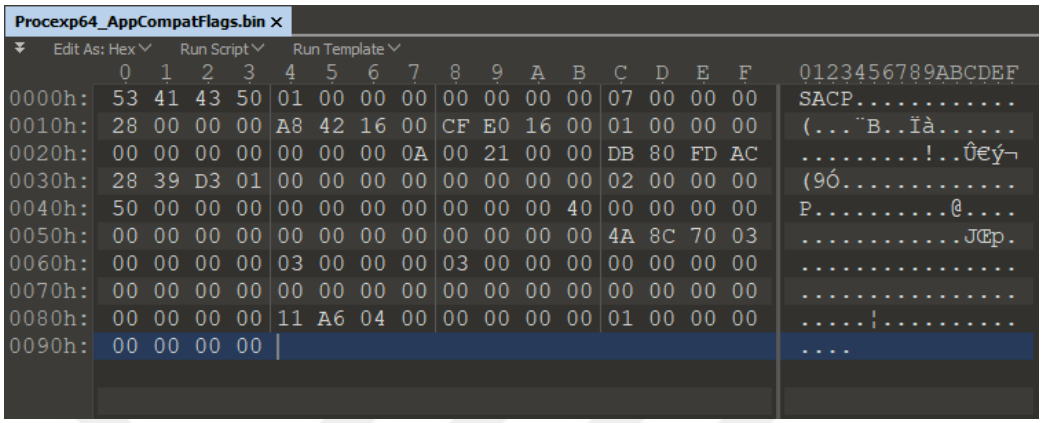


Figure 4.9 is example of 148 bytes length value:

Figure 4.9: "Compatibility Assistant\Store" registry key - 148 bytes sample



Following research methodology were followed:

i. Take snapshot of "AppcompatFlags\Compatibility Assistant\Store" registry key
ii. Launch program
iii. Take another snapshot of registry "AppcompatFlags\Compatibility Assistant\Store" key.
iv. Compare values

Above process was repeated several times but changing one of the following variables while fixing others:

a. Execution time
b. Executable file modification date
c. Executable file creation date
d. Program file name
e. Program file path
f. Using same program but compiled in different architecture 32 vs 64.
g. Executing same program several times

After examination of values' binary data, several forensically important items were extracted.

First 60 bytes section which is included in all three groups (60, 108, 148) are explained in Table 4.4.

Table 4.4: "Compatibility Assistant\Store" value's data first 60 bytes section structure

Offset	Size	Description
0x0000	4	Header of structure. Always set to "SACP" in ASCII
0x0004	4	Observed static value "01 00 00 00"
0x0008	4	Observed static value "00 00 00 00"
0x000C	4	Observed static value "07 00 00 00"
0x0010	4	Observed static value "28 00 00 00"
0x0014	4	File size in bytes
0x0018	4	Unknown variable value
0x001C	4	Observed static value "01 00 00 00"
0x0020	4	Observed static value "00 00 00 00"
0x0024	8	Unknown variable value
0x002C	8	Observed static value "DB 80 FD AC 28 39 D3 01"
0x0034	8	Observed static value "00 00 00 00 00 00 00 00"

Second section, included in two groups (108, 148), is described in Table 4.5.

Table 4.5: "Compatibility Assistant\Store" value's data second section structure

Offset	Size	Description
0x003C	4	Observed static value "02 00 00 00"
0x0040	4	Next structure size in bytes.
0x0044	4	Observed static value "00 00 00 00"

0x0048	8	Observed static values "00 00 00 40 10 00 00 00" and "00 00 00 00 00 00 00 00"
0x0050	12	Observed static values "00 00 00 00 00 00 00 00 00 00 00 00"
0x005C	4	Variable value that is unique per executable file name and location.
0x0060	4	Observed static value "00 00 00 00"
0x0064	4	Run count
0x0068	4	Unknown variable value

Third section is only included in last group (148). To verify findings of this section, more samples need to be generated. This section of variable size, the exact size can be obtained by checking integer value starting at offset 0x0040

Last section size = int_value(0x0040 to 0x0044) - 40;

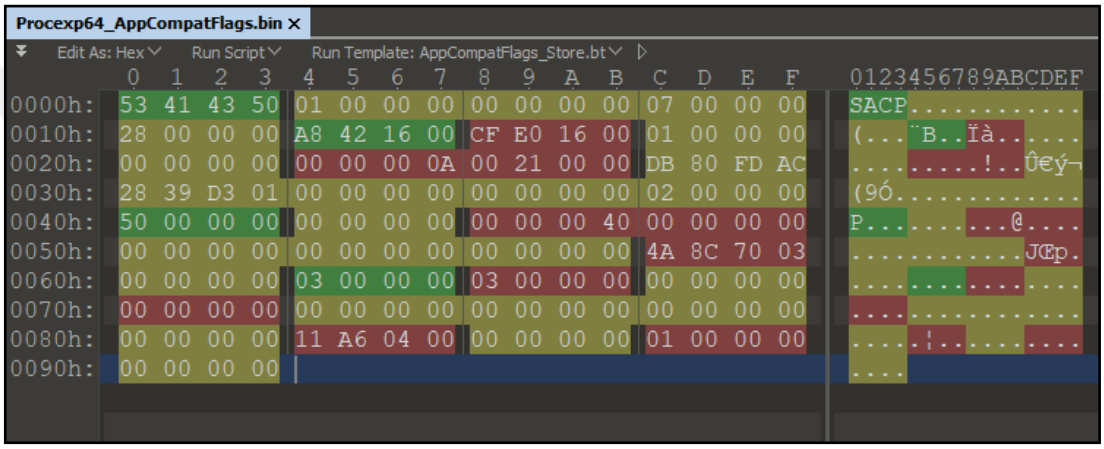
This section doesn't seem to contain forensically important items that are being studied in this research as shown in Table 4.6. To verify findings, more samples need to be studied. However, it was hard to generate more samples, since the reason that Windows operating system generates this section to only few programs is still unknown.

Table 4.6: "Compatibility Assistant\Store" value's data last section structure

Offset	Size	Description
0x006C	4	Observed static value "00 00 00 00"
0x0070	4	Unknown variable value
0x0074	16	Observed static value "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00"
0x0084	4	Unknown variable value
0x0088	4	Observed static value "00 00 00 00"
0x008C	4	Unknown variable value
0x0090	4	Observed static value "00 00 00 00"

Figure 4.10 is visual presentation of one sample binary value after some important data was extracted:

Figure 4.10: Binary data structure of sample value of "Compatibility Assistant\Store" registry key



Where:

a. **Yellow**: Represent static values. Static values are often doesn't have forensically valuable information as they don't uniquely identify information about executed program.
b. **Green**: Known extracted information
c. **Red**: Unknown variable values. Those values are changing among different executed programs. However, it wasn't observed that they contain any of the important forensically important execution items.

4.1.1.2 Findings summary

Analysis results of "AppCommpatFlags\Compatibility Assistant\Store" are in Table 4.7.

Table 4.7: "Compatibility Assistant\Store" findings summary

Item	"Compatibility Assistant\ Store"
Executable File Name	YES
Executable File Path	YES
Executable File Size	YES
Executable File Date	NO
Executable File Hash	NO
Execution Date	NO
Execution History	NO
Executions Count	YES
User	YES
Requires Reboot?	NO

4.1.1.3 Parsing tools results

Results of running different parsing tools against "AppCommpatFlags\Compatibility Assistant\Store" are listed in Table 4.8.

Table 4.8: "Compatibility Assistant\Store" parsing tools' results

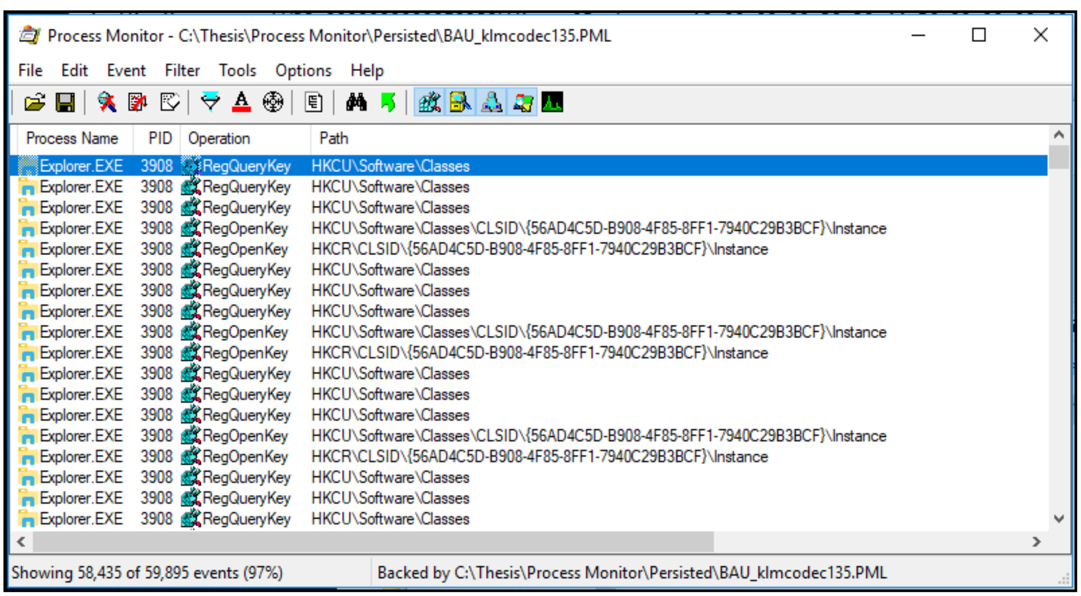
Item	"Compatibility Assistant\Store"
AXIOM	NO
RegRipper	Partial (Only executable path)
Registry Explorer	NO
TZWorks	NO

4.1.2 AppCompatFlags - Persisted

4.1.2.1 Analysis

"PCA stores the list of all programs for which it came up under the following key for each user, even if no compatibility modes were applied HKCU\Software\Microsoft\Windows NT\Current Version\AppCompatFlags\Compatibility Assistant\Persisted" (Hameed, 2007). However, on newer version of Windows, Windows 8 and above, "Persisted" key is no longer used, and data is stored under "HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Compatibility Assistant\Store" instead (Harrell, 2013). To verify these findings, some programs were executed on Windows 10 system and access and modifications on “AppCompatFlags\Compatibility Assistant\Persisted” were monitored with Process Explorer:

Figure 4.11: No access to "Compatibility Assistant\Persisted" registry key during process creation



No access or modifications of "Persisted" key were found as it shows in Figure 4.11. However, on another Windows 10 systems that have been updated from previous Windows

8.1 system, "AppCompatFlags\Compatibility Assistant\Persisted" were found in "HKCU" and "HKLM" and populated with several values as it shows in Figure 4.12 and Figure 4.13.

Figure 4.12: "Compatibility Assistant\Persisted" registry key contents on Windows 10 updated from older version

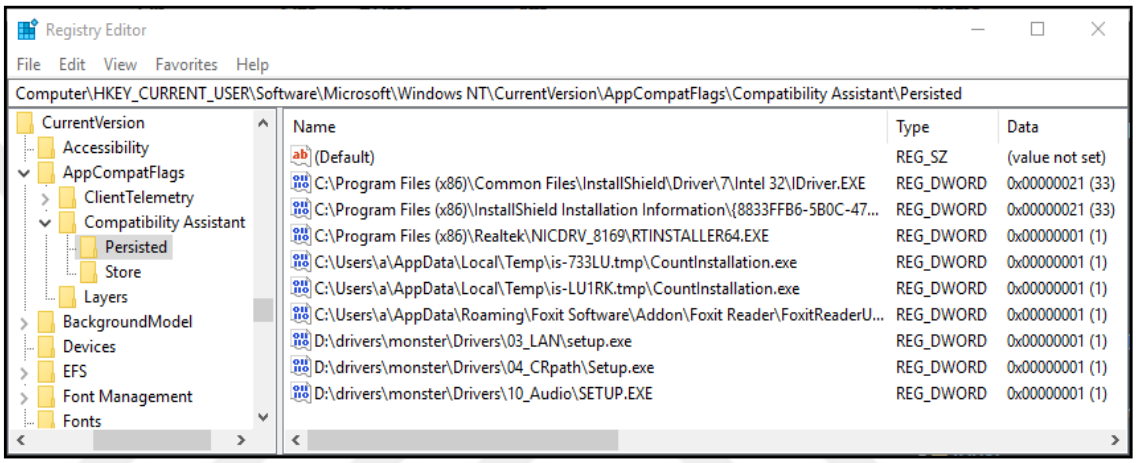
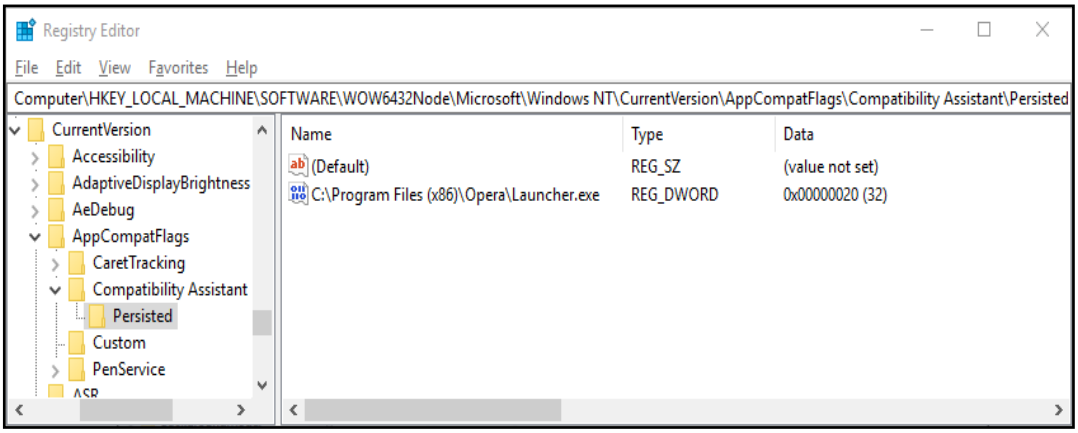


Figure 4.13: Another sample of "Compatibility Assistant\Persisted" registry key contents under HKLM



Some of those programs were extracted from Windows 10 system, which has been upgraded from Windows 8.1, and executed on Windows 10, which was freshly installed,

and again no access or modification were noted on "AppCompatFlags\Compatibility Assistant\Persisted" key.

The values under "AppCompatFlags\Compatibility Assistant\Persisted" are of type registry "REG_DWORD" values, unlink values under new "AppCompatFlags\Compatibility Assistant\Store" key which are "REG_BINARY" which save binary data. This means, values under "Persisted" key don’t save as much information as values under "Store" keys. Observed values are in Table 4.9.

Table 4.9: Programs count based on "Persisted" value

Value	Programs count
1	7
32	1
33	2

Exact value meaning is not known. However, according to a web page (sevenforums, 2010), they maybe indicate compatibility issues and whether program has run successfully or not.

4.1.2.2 Findings summary

Analysis results of "AppCommpatFlags\Compatibility Assistant\Persisted" are in Table 4.10.

Table 4.10: "Compatibility Assistant\Persisted" findings summary

Item	"Compatibility Assistant\Persisted"
Executable File Name	YES
Executable File Path	YES

Executable File Size	NO
Executable File Date	NO
Executable File Hash	NO
Execution Date	NO
Execution History	NO
Executions Count	NO
User	YES (in case saved in NTUSER.DAT)
Requires Reboot?	NO

4.1.2.3 Parsing tools results

Results of running different parsing tools against "AppCommpatFlags\Compatibility Assistant\Persisted" are listed in Table 4.11.

Table 4.11: "Compatibility Assistant\Persisted" parsing tools' results

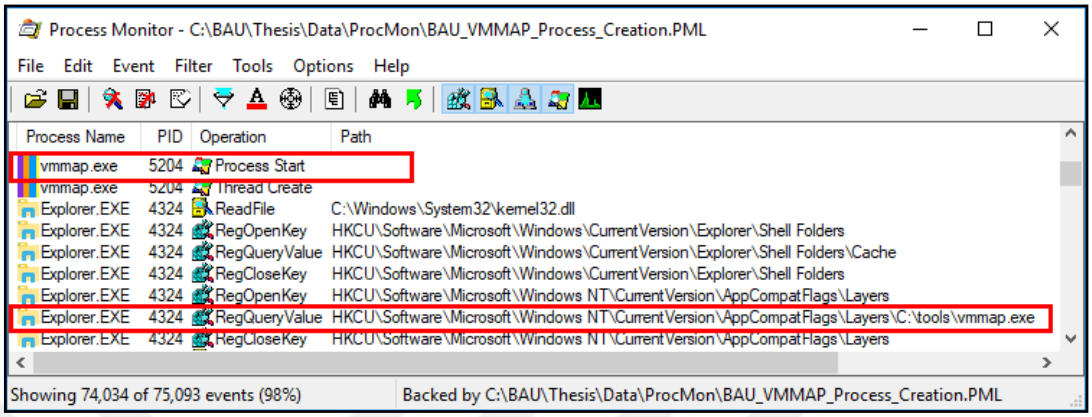
Item	"Compatibility Assistant\Persisted"
AXIOM	NO
RegRipper	Partial (Only executable path)
Registry Explorer	NO
TZWorks	NO

4.1.3 AppCompatFlags - Layers

4.1.3.1 Analysis

During research on process creation, it was found that in addition to "AppCompatFlags\Compatibility Assistant\Store" key, another registry key "AppCompatFlags\Layers" key is being checked as it appears in Figure 4.14:

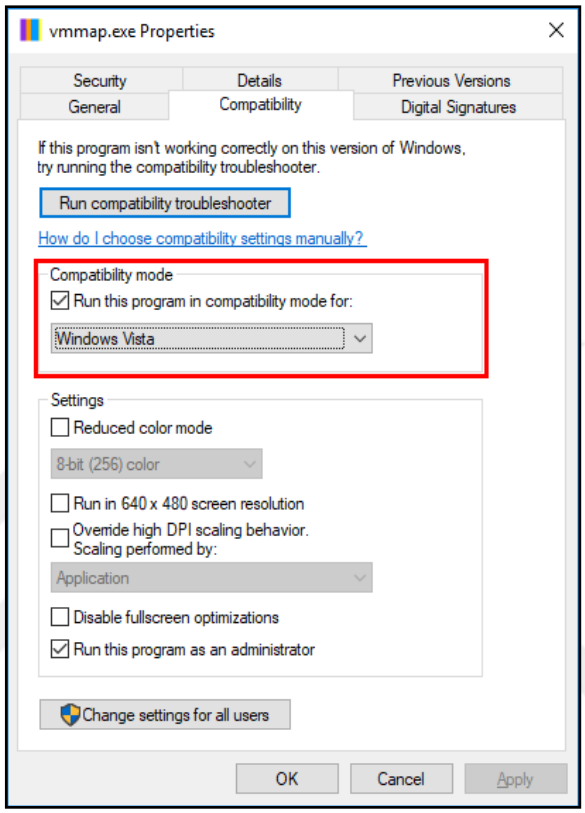
Figure 4.14: Explorer.exe queries "Layers" key on process creation



"AppCompatFlags\Layers" key is related to Microsoft Windows compatibility mode (Verboon, 2011). The main use of compatibility mode is to enable older programs, that have issues with running on newer Windows version, to run smoothly and without programs on newer Windows versions (Microsoft, 2018).

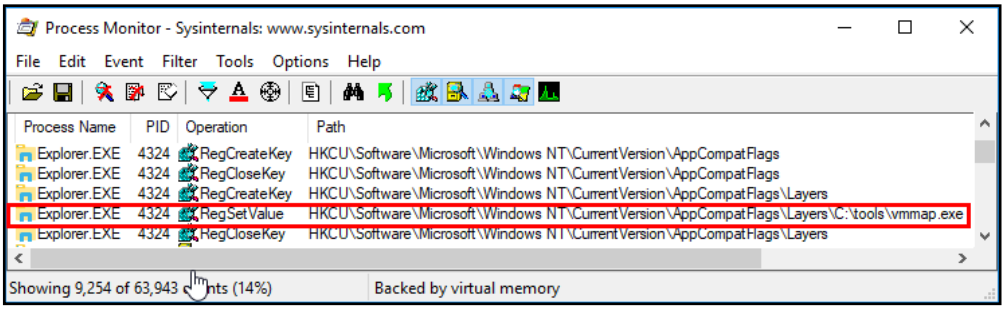
To verify above findings, compatibility mode for one program was set to run on Windows Vista as it shows in Figure 4.15:

Figure 4.15: Compatibility mode for executable modules



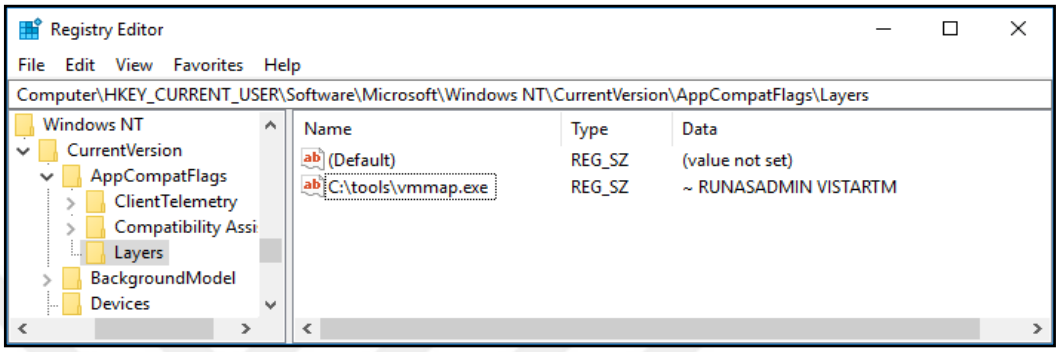
Access to registry was monitored during compatibility mode changing process.

Figure 4.16: Explorer.exe creates new value under "Layers" key on process creation



As Figure 4.16 shows, a new value under "AppCompatFlags\Layers" was set.

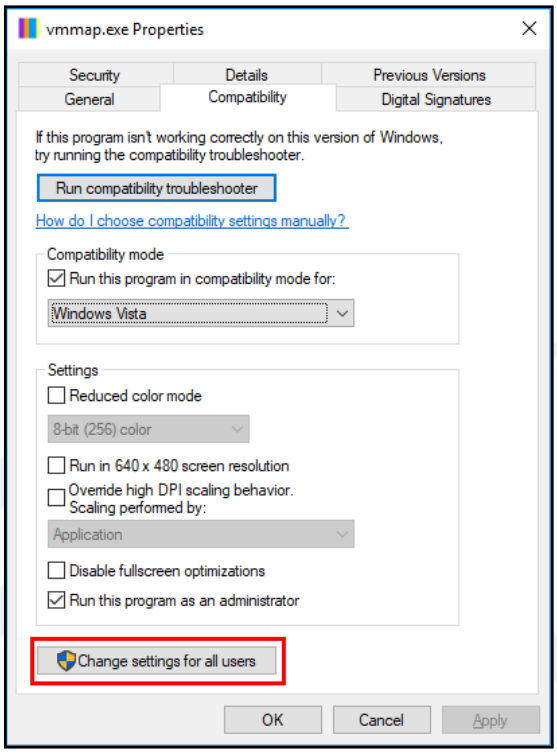
Figure 4.17: "Layers" registry key contents



The new value name is the program path: "C:\tools\vmmap.exe" as shown in Figure 4.17. The value of type "REG_SZ" which saves text data. The value's data is compatibility mode parameters.

Compatibility mode can also be set to all users on system as opposed to current user as demonstrated in Figure 4.18:

Figure 4.18: Change compatibility mode for all users



This is important difference because in this case, "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers" key will change and it won't be possible to find user who run program by this artifact on its own.

It worth note however, that this artifact by its own doesn't guarantee that program was executed. However, it shows user knowledge of program and it can be used in correlation with other artifacts to understand program execution behavior.

4.1.3.2 Findings summary

Analysis results of "AppCommpatFlags\Layers" are in Table 4.12.

Table 4.12: "AppCompatFlags\Layers" findings summary

Item	"AppCompatFlags\Layers"
Executable File Name	YES
Executable File Path	YES
Executable File Size	NO
Executable File Date	NO
Executable File Hash	NO
Execution Date	NO
Execution History	NO
Executions Count	NO
User	YES (in case saved in NTUSER.DAT)
Requires Reboot?	NO

4.1.3.3 Parsing tools results

Results of running different parsing tools against "AppCommpatFlags\Layers" are listed in Table 4.13.

Table 4.13: "AppCompatFlags\Layers" parsing tools' results

Item	"AppCompatFlags\Layers"
AXIOM	NO
RegRipper	NO
Registry Explorer	NO
TZWorks	NO

4.1.4 AppCompatCache (ShimCache)

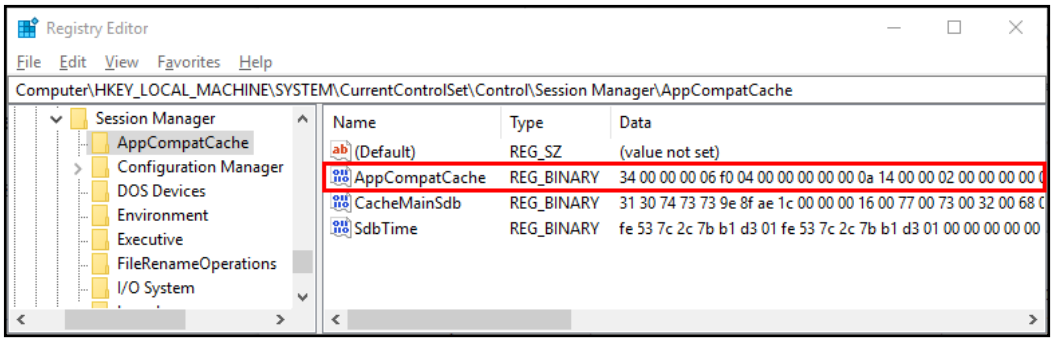
4.1.4.1 Analysis

Microsoft Windows Application Compatibility Infrastructure (Shim Infrastructure) is a technology used by Windows to allow older applications to run on newer Windows versions without having compatibility issues (Microsoft.com, 2012). Shim Cache is one component of Shim Engine and it's being used to cache various information about executable modules (Ionescu, 2007). The information stored in Shim Cache differs significantly from one Window version to another and incident responders should be aware of those differences in order to accurately understand what had happen.

The detailed implementation of Shim Cache is beyond the scope of this research. However, it's important for incident responder to understand that Windows stores Shim Cache in memory and then it serializes Shim Cache to registry on system shutdown or reboot (Davis, 2012). Thus, during live system analysis, checking Shim Cache registry values will not shows executables from last system boot. To solve this issue, Mandiant has developed a volatility module to extract Shim Cache information from memory which allows incident responders to check executable modules from last system boot as well (Mandiant, 2017).

The registry value which is used to store Shim Cache is "AppCompatCache". The value is located under "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\AppCompatCache" key as shown in Figure 4.19.

Figure 4.19: "AppCompatCache" registry key contents

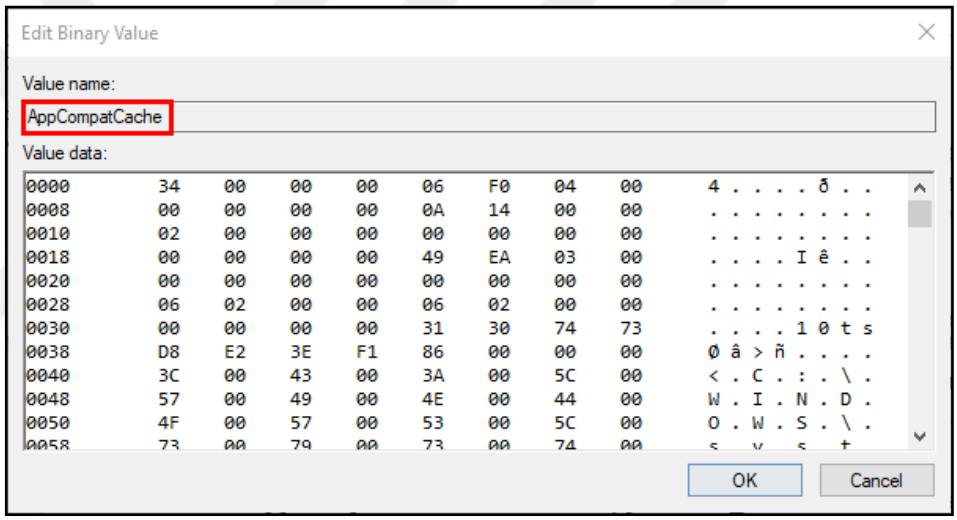


"AppCompatCache" value's location in registry indicates two important pieces of information for incident responders: First, since the value is under "HKLM", Shim Cache

can’t be used to determine user account who launched\viewed the executable. Second, the value is under "CurrentControlSet", which means in order for incident responders to extract information about all executable modules on system, they have to check all ControlSets presented in registry (Carvey, 2016).

"AppCompatCache" value is of type "REG_BINARY" and it stores binary data as shown in Figure 4.20:

Figure 4.20: "AppCompatCache" value's data



In order to understand data stored in binary data, group of executable modules were tested on Windows system. Executables were divided into three groups:

a. Group 1: Executable modules that will be executed on system.
b. Group 2: Executable modules that will not be executed but will be viewed in Windows Explorer
c. Group 3: Executable modules that will be copied to machine from external source but will not be executed or viewed on the system.
d. Group 4: Executable modules that will be downloaded on system but will not be executed or viewed by Windows Explorer.

Each group contains four executable modules as demonstrated in Table 4.14.

Table 4.14: Executables groups information

Group	Executable Name	Architecture	Size (Bytes)
Group 1	mozillahistoryview.exe	64x	144,080
Group 1	regscanner.exe	32x	60,112
Group 1	uninstallview.exe	32x	129,744
Group 1	usbdeview.exe	32x	130,768
Group 2	chromecacheview.exe	32x	70,352
Group 2	dnsquerysniffer.exe	32x	171,216
Group 2	hashmyfiles.exe	64x	59,088
Group 2	jumplistsview.exe	32x	90,320
Group 3	awatch.exe	32x	35,328
Group 3	controlmymonitor.exe	32x	111,824
Group 3	credentialsfileview.exe	64x	155,856
Group 3	timezonesview.exe	32x	85,712
Group 4	foldertimeupdate.exe	64x	124,416
Group 4	lastactivityview.exe	32x	134,864
Group 4	openedfilesview.exe	32x	71,376
Group 4	whoistd.exe	32x	59,088

Following methodology is used:

i. Save AppCompatCache registry value binary data.

ii. Do one group specific test at a time.

iii. Shutdown system to allow Shim Cache serialization to registry.

iv. Boot the system.

v. Save a second copy of AppCompatCache registry value binary data.
vi. Compare the saved two value's binary data on binary level to check for new entries.
vii. Repeat steps#i to vi but with different group test at step#ii:
    a. Launch executable modules
    b. View executable modules with Windows Explorer
    c. Copy executable module from external source
    d. Download executable module from internet.

Results are in Table 4.15.

Table 4.15: AppCompatCaches entries creation results

Group	Description	AppCompatCache entries
Group 1	Executed	4 Entries
Group 2	Viewed by Windows Explorer but not executed	4 Entries
Group 3	Copied from external drive. Not viewed and not executed	No entries
Group 4	Downloaded from internet. Not viewed and not executed	No entries

Results show that opening a folder with Windows Explorer creates entries in AppCompatCache value. So, "AppCompatCache" doesn't guarantee program execution by itself. However, it's at least showing the presence of executable on system and possible user knowledge of that executable.

"AppCompatCache" value's data is in binary format and it needs to be reversed engineered to extract human readable information out of it. In order to reverse-engineer "AppCompatCache" binary structure, many tests with different executables were performed, and testing methodology was as following

i. Generate Shim Cache entry by executing or using Window Explorer to view executable module.
ii. Shutdown system to serialize Shim Cache to "AppCompatCache" value.

iii. Extract "AppCompatCache" value’s data.

iv. Use Hex Editor to check binary data

v. Modify some executable parameters and redo steps#i to iv.

Several important items were identified inside binary structure, while some other values inside binary structure remain unknown. The unknown values are not likely to contain items of forensics interest covered in this research. The binary structure contains a header section and series of entries, where each entry seems to belong to one executable module.

4.1.4.1.1 Header section

Contains information such as first entry offset and number of entries. Header section is described in Table 4.16.

Table 4.16: AppCompatCache value's data header section structure

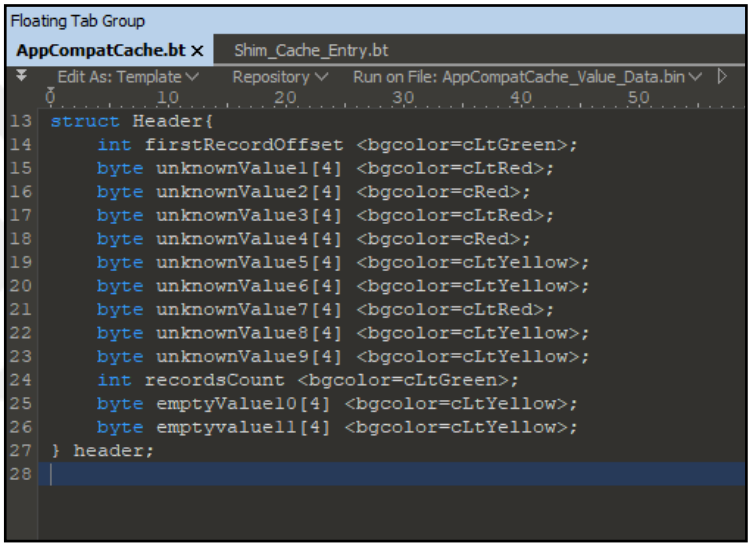
Offset	Size	Description
0x0000	4	Offset to first record. On Windows-10 1079, value =0x00000034 (52) bytes
0x0004	4	Unknown variable value
0x0008	4	Unknown variable value
0x000C	4	Unknown variable value
0x0010	4	Unknown variable value
0x0014	8	Observed static value "00 00 00 00 00 00 00 00"
0x001C	4	Unknown variable value
0x0020	8	Observed static value "00 00 00 00 00 00 00 00"
0x0028	4	Entries count
0x002C	8	Observed static value "00 00 00 00 00 00 00 00"

During this research, entries count was found to be inconsistent with actual entries count stored "AppCompatCache" in last version of Microsoft Windows 10 during the time of this writing: Windows 10 1097. So, forensics tools that parse this artifact must not rely on this

field. Rather, forensics tools should scan all binary data and find all entries by searching for entry signature "10ts".

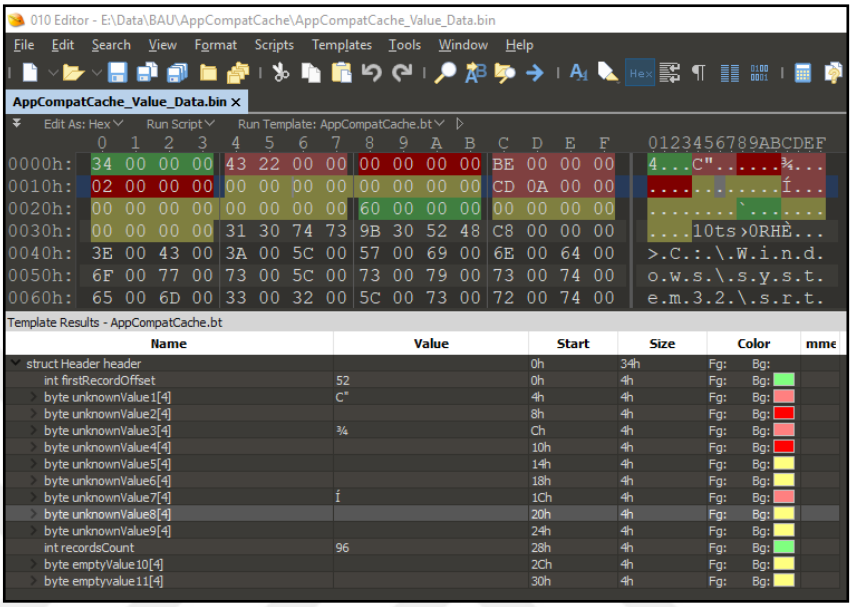
Figure 4.21 shows a simple script that was developed to extract information in header section:

Figure 4.21: "AppCompatCache" header section parsing script



Results of running script are shown in Figure 4.22:

Figure 4.22: Parsing results of "AppCompatCache" header section



4.1.4.1.2 Entry Section

Shim Cache entries' binary structure has changed significantly among different versions of Windows (Davis, 2012). In this paper, Shim Cache entry binary structure that is used on latest version of Windows, Windows 10 1097, has been examined.

In order to reverse-engineer Shim Cache entry structure, following methodology was followed:

i. Extract Shim Cache entries that belongs to specific executable under study from AppCompatCache value.
ii. Do Shim Cache entry specific test
iii. Shutdown system to allow Shim Cache serialization to registry.
iv. Boot the system.
v. Save a second copy of same Shim Cache entries from step#i.
vi. Compare the saved two copies of Shim Cache entries binary level.
vii. Repeat steps#i to vi but with Shim Cache test at step#ii:

a. Launch same executable module more than one time
b. Change location of executable module on file system.
c. Replace executable that has been already shimmed with different executable but same and path.

Binary structure of "AppComaptCache" entry is described in Table 4.17.

Table 4.17: "AppCompatCache" entry's data structure

Offset	Size	Description
0x0000	4	Signature. On Windows 10 1097, value is "10ts"
0x0004	4	CRC32 value of entries data from 0x000C till entry's end
0x0008	4	Entry's data section size.
0x000C	2	Path length
0x0014	..	Path stored in UTF-16 little endian
..	8	FILETIME value of executable file last modified time stored in $STANDARD_INFORMATION.
..	4	Internal blob section size
..	end	Internal blob section data

4.1.4.1.3 Internal Blob Section

Internal binary blob structure is still unknown. However, it doesn't seem to contain any forensics items of interest studied in this paper except for program execution. While current forensics tools, that parse AppCompatCache on Windows 10, are not able to tell if executable module is executed or it just merely viewed by Windows Explorer, research done during preparing this paper found that executed programs have additional binary data section added to internal binary blob.

During this research, it was found that if executable module has been launched, it's internal binary blob size was > 100 bytes and additional section of size 36 bytes is added. Binary blob structure is described in Table 4.18.

Table 4.18: "AppCompatCache" entry internal blob's data structure

Offset	Size	Description
0x0000	4	Internal blob section size
0x0004	..	Blob binary data
..	36	If blob size > 100 bytes, this section is added. Observed value: "40 00 00 00 04 00 00 00 01 00 00 00 20 00 00 00 04 00 00 00 00 00 00 00 00 01 00 00 04 00 00 00 01 00 00 00"

Table 4.19 shows Shim Cache entries for executed and not executed executable modules:

Table 4.19: Internal blob size for executed and not executed programs

Executable name	Executed	Internal binary blob size (bytes)
chromecacheview.exe	No	100
dnsquerysniffer.exe	No	100
hashmyfiles.exe	No	100
jumplistsview.exe	No	100
mozillahistoryview.exe	Yes	238
regscanner.exe	Yes	148
uninstallview.exe	Yes	660
usbdeview.exe	Yes	136

Below is comparison between Shim Cache entry of same executable module before and after execution.

Figure 4.23 shows entry data before execution:

Figure 4.23: "AppCompatCache" entry data before execution

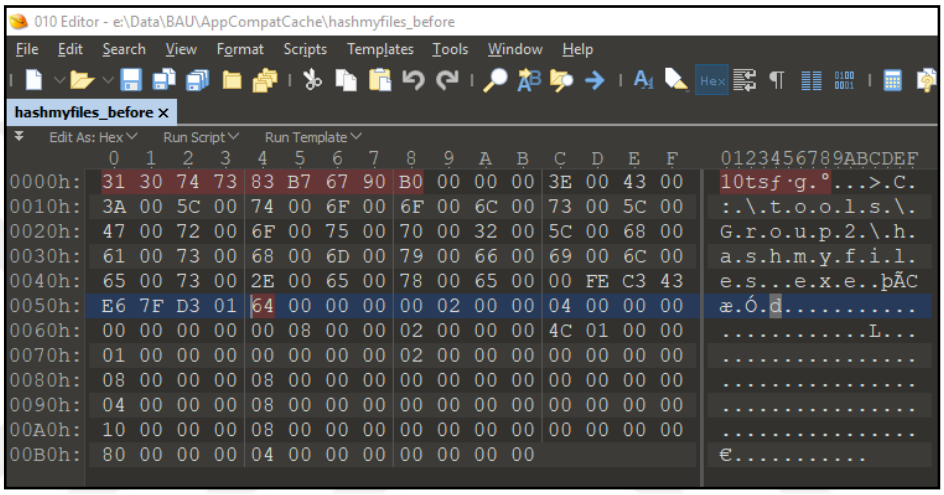
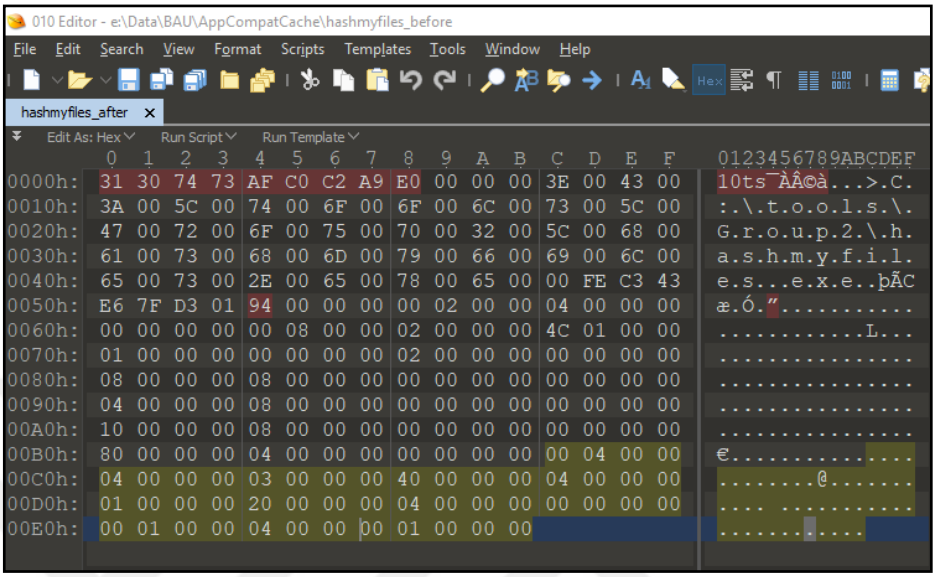


Figure 4.24 show entry data after Execution:

Figure 4.24: "AppCompatCache" entry data after execution

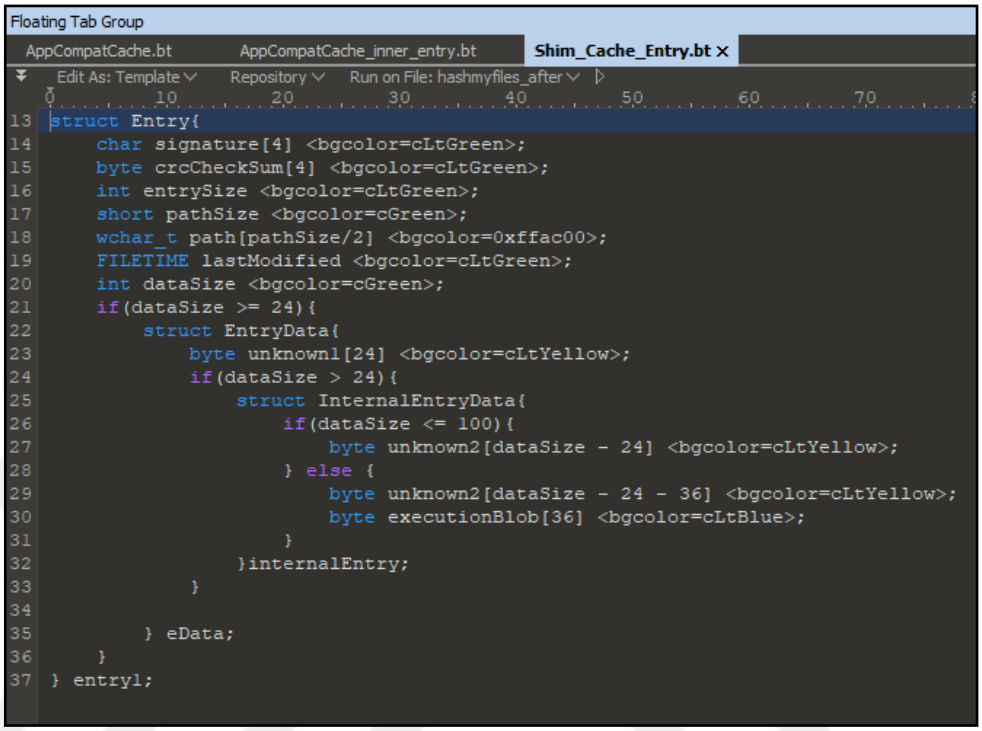


Different data marked with red, and added data marked with yellow.

Above figures show that only CRC-32 and data's size values are changed, and additional section is added when executable module is launched.

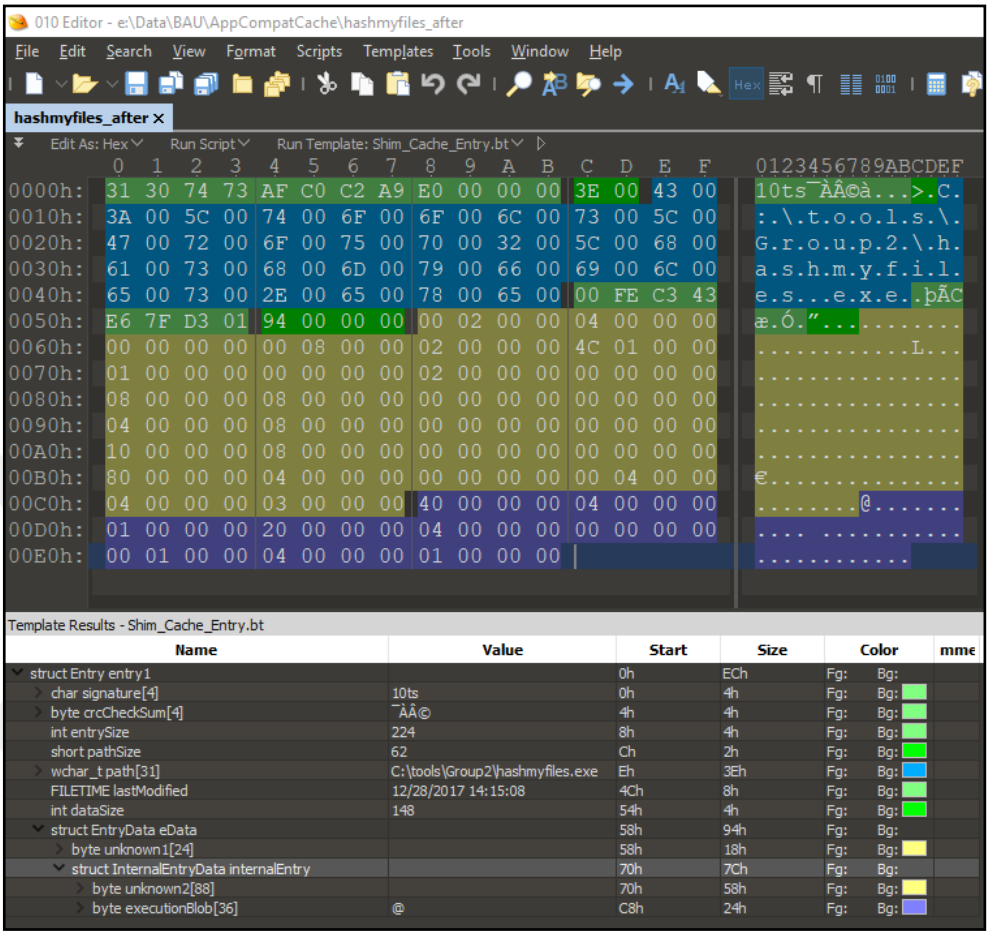
As a result of this research, a script was developed to parse Shim Cache entry data. The script is shown in Figure 4.25.

Figure 4.25: "AppCompatCache" entry section parsing script



Results of running this script, on Shim Cache entry which belongs to executable module that has been executed, are shown in Figure 4.26:

Figure 4.26: "AppCompatCache" entry data parsing results



4.1.4.2 Execution date

During research, it was found that new Shim Cache entries are always added after header section. So, while there are no execution date values stored in "AppCompatCache" values, entries order, in "AppCompatCache" value, along with "AppCompatCache" registry key's last update date, can be used to determine executable modules order of execution.

4.1.4.3 Findings summary

Analysis results of "AppCompatCache" are in Table 4.20.

Table 4.20: "AppCompatCache" findings summary

Item	AppcompatCache
Executable File Name	YES
Executable File Path	YES
Executable File Size	NO
Executable File Date	NO
Executable File Hash	NO
Execution Date	NO
Execution History	NO
Executions Count	NO
User	NO
Requires Reboot?	YES

4.1.4.4 Parsing tools results

Results of running different parsing tools against "AppCompatCache" are listed in Table 4.21.

Table 4.21: "AppCompatCache" parsing tools' results

Item	AppCompatCache
AXIOM	NO
RegRipper	YES
Registry Explorer	YES
TZWorks	YES

4.2 BACKGROUND ACTIVITY MODERATOR (BAM)

4.2.1 Analysis

During this research, another Windows registry key was found to be checked on process creation and process exit as shown in Figure 4.27.

Figure 4.27: Explorer.exe queries "bam\UserSettings\%User Security Identifier%\" key on process creation

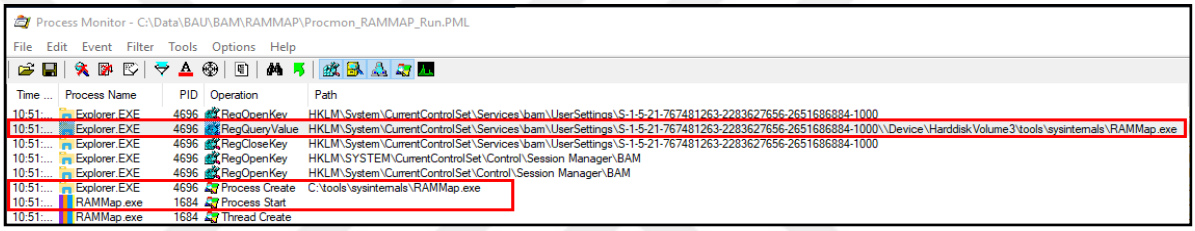


Figure 4.27 shows that "Explorer.exe", which is parent process of "RAMMAP.exe" process, checks for registry value: "HKLM\System\CurrentControlSet\Services\bam\UserSettings\S-1-5-21-767481263-2283627656-2651686884-1000\Device\Harddisk\Volume3\tools\sysinternals\RAMMap.exe" before "Process Start" event. The location of this registry value is under "Services" key which contains a configuration information of services on Windows system. Also, this value location contains "Device\Harddisk\Volume3\tools\sysinternals\RAMMap.exe", which is the name and the full path of the executable module being executed, and "S-1-5-21-767481263-2283627656-2651686884-1000" is the User Security Identifier (SID) of user account who executed "RAMMAP.exe". The location is also under "CurrentControlSet", so it's important for incident responder to check all available control sets on system being examined in order to extract "bam" information from other control sets too.

Figure 4.28: RAMMap.exe queries "bam\UserSettings" key on process exit

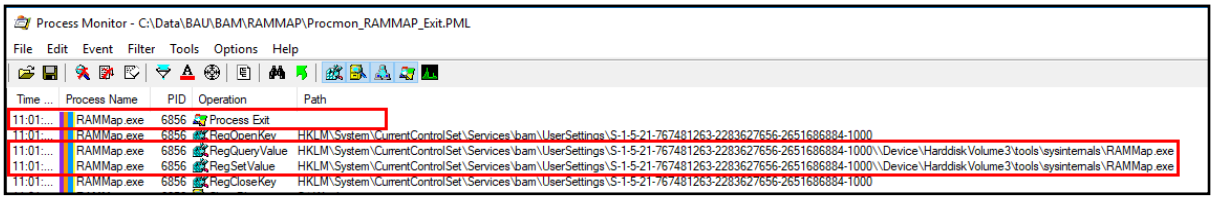
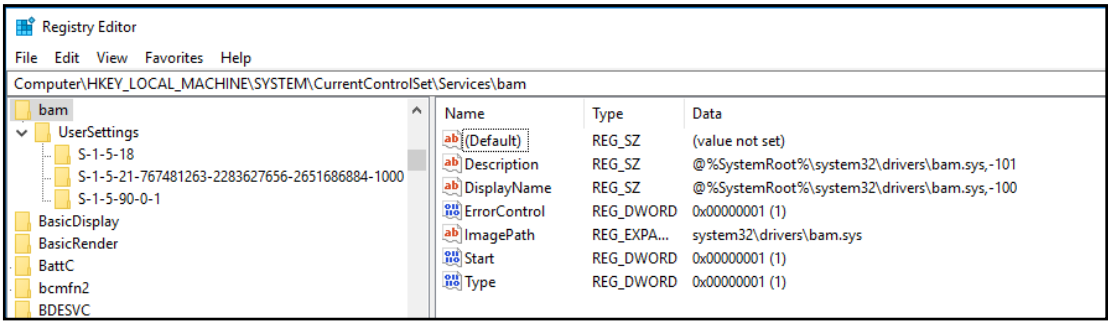


Figure 4.28 is for process exit "RAMMAP.exe", and it shows that "HKLM\System\CurrentControlSet\Services\bam\UserSettings\S-1-5-21-767481263-2283627656-2651686884-1000\Device\HarddiskVolume3\tools\sysinternals\RAMMap.exe" registry value is being checked and changed again after "Process Exit" event.

As noted earlier, location of registry value is under "Services" key, which contains a list of services installed on Windows system, and subkey name is "bam". Configuration information of "bam" are shown in Figure 4.29.

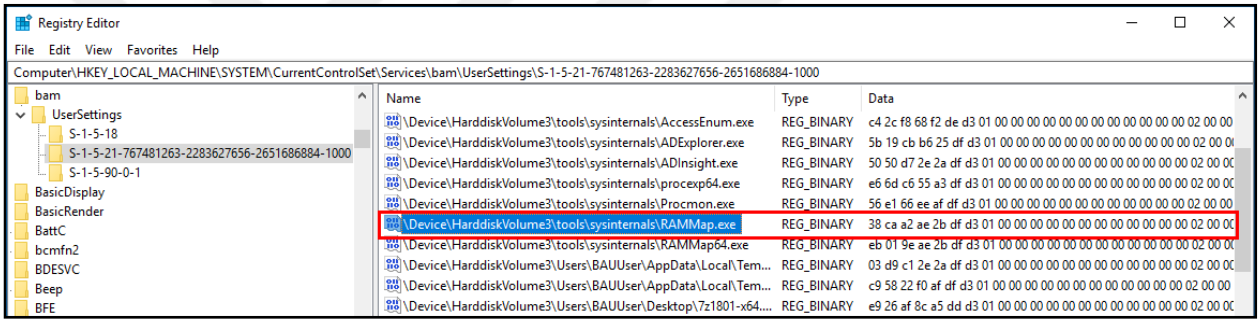
Figure 4.29: "bam" registry key contents



"bam" service is using "bam.sys" which is a kernel mode driver that was introduced only in Windows 10 version 1079 which is the latest version of Windows at the time of this writing (batcmd.com, 2017). The "Type" value is "0x00000001" which indicates that it's kernel driver service (MSDN, n.d.). The "Start" type of service is "0x00000001" which means that this service is being loaded by "IO Subsystem". The exact purpose of

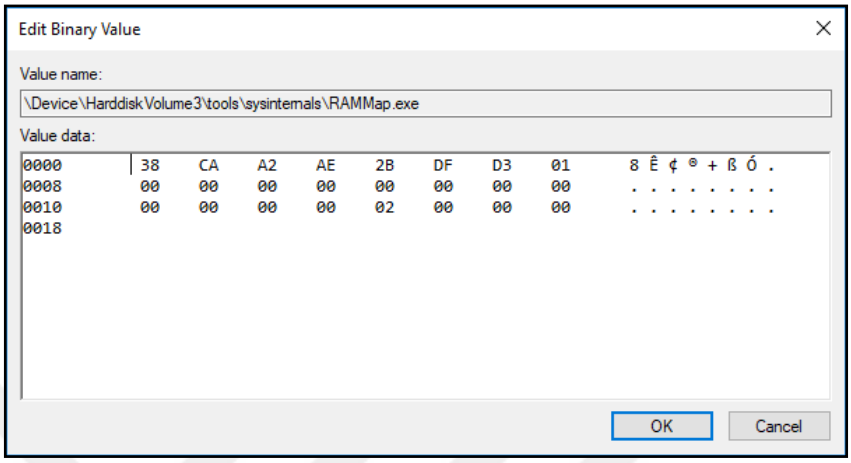
"Background Activity Moderator" is still unknown, however, it can to be related to "Desktop Activity Moderator" which is designed to enhance battery life (MSDN, 2017). "Background Activity Moderator" (BAM) service has subkey "UserSettings" and under that subkey there is a list of User Security Identifier (SIDs) of users on system. This will help incident responders with determining user account who executed the application. Under "User Security Identifier" key, there is a list of values that correspond to executed programs as shown in Figure 4.30.

Figure 4.30: List of "bam" entries



Where each value corresponds to one executable module which has been executed, except for "SequenceNumber" and "Version" values. The value name contains full path of executable module which has be executed. The values are of type 'REG_BINARY" which indicates that stored data is in binary format. Figure 4.31 shows values data content for sample value

Figure 4.31: Sample "bam" entry value data



Above figure shows a "bam" entry sample of "RAMMAP.exe" executable module. All "bam" entries values data is of length 24 bytes. Binary data needs reverse-engineering to extract useful information. For that purpose, the following methodology was followed:

i. Save registry values under "bam\UserSettings\%User Security Identifier%".
ii. Launch executable module.
iii. Save a second copy of registry values under "bam\UserSettings\%User Security Identifier%".
iv. Compare values at binary level.
v. Repeat steps #i to iv with different executable modules.

Binary structure of "bam" value is described in Table 4.22.

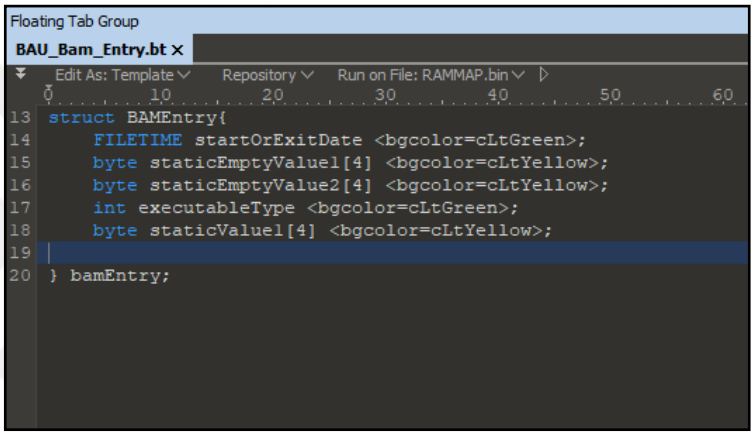
Table 4.22: "bam" value data structure

Offset	Size	Description
0x0000	8	Execution or exit time of executable module stored in FILETIME format.
0x0008	4	Static empty value "00 00 00 00"
0x000C	4	Static empty value "00 00 00 00"

0x0010	4	Application type. For Standard Windows executables, value is "00 00 00 00". For Universal Windows App or Store App, the value is "00 00 00 01"
0x0014	4	Static value. Observed value "00 00 00 02"

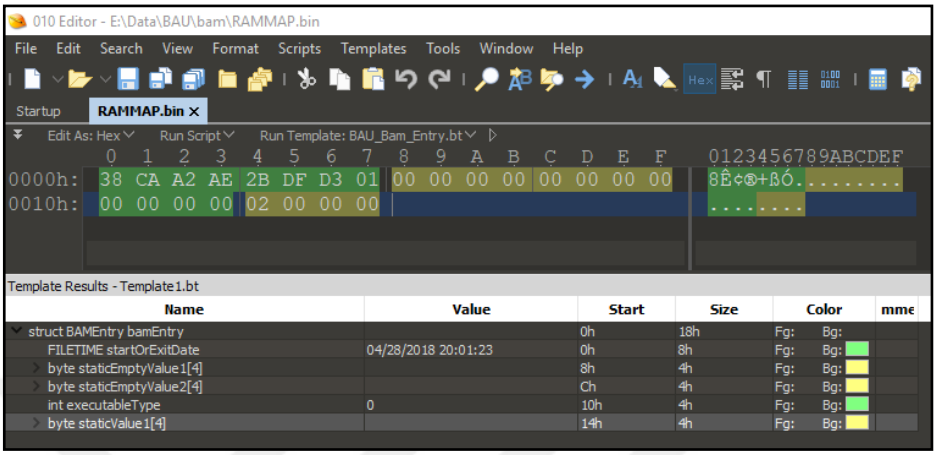
Figure 4.32 shows a script that was developed to parse "bam" binary structure:

Figure 4.32: "bam" entry parsing script



Parsing results are demonstrated in Figure 4.33.

Figure 4.33: "bam" entry parsing results



Some researchers mention that that "bam" entries are getting updated on system shutdown or boot (Katsavounidis, 2018). However, in this research, this claim was found to be not accurate, as there was no evidence of "bam" entries update on either system update or shutdown. "bam" entry is getting set directly upon program execution and again "bam" entry getting updated when program is closed. The value that is changed, in "bam" entry upon program close in the filetime value at offset 0x0000 of value data.

4.2.2 Findings Summary

Analysis results of "bam" are in Table 4.23.

Table 4.23: "bam" findings summary

Item	"bam"
Executable File Name	YES
Executable File Path	YES
Executable File Size	NO
Executable File Date	NO
Executable File Hash	NO
Execution Date	YES

Execution History	NO
Executions Count	NO
User	YES
Requires Reboot?	NO

4.2.3 Parsing Tools Results

Results of running different parsing tools against "bam" are listed in Table 4.24.

Table 4.24: "bam" parsing tools' results

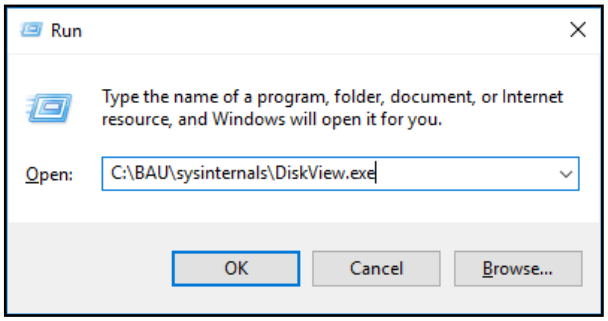
Item	"bam"
AXIOM	NO
RegRipper	YES
Registry Explorer	YES
TZWorks	NO

4.3 RUN COMMAND

4.3.1 Analysis

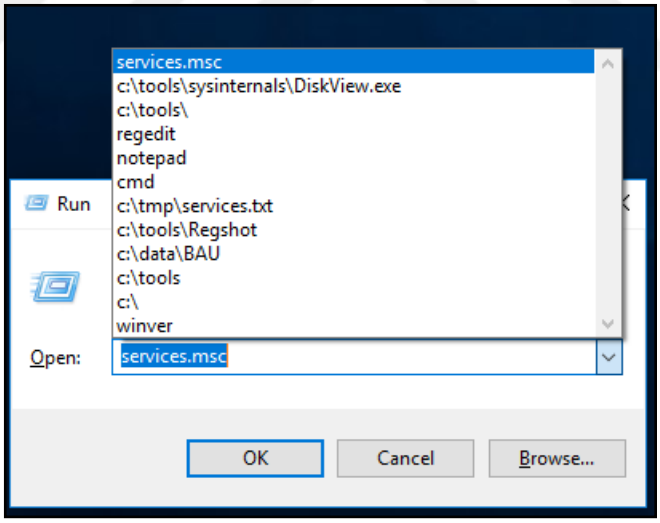
Windows "Run" command is used to open a file, a folder or to execute an application. The user needs to provide full path of the executable or just the executable’s name in case that executable located in one of directories listed in "Path" environment variable. Windows "Run" command is shown in Figure 4.34.

Figure 4.34: "Run" command dialog



Windows stores a list of recent "Run" commands, as shown in Figure 4.35, and this feature can be used in programs execution incidents response (Carvey, 2013).

Figure 4.35: Recent "Run" commands



Like with many other Windows recent items, Windows stores recent "Run" commands in "Most Recently Used List" structure in Registry. The key location is "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU". The presence of "RunMRU" under "HKCU", means that "RunMRU" can be used to show the user account who executed the "Run" command.

During this research, it was found that new "Run" commands are added to this key only after the program is executed.

Figure 4.36: Explorer.exe creates new value under "Explorer\RunMRU" key

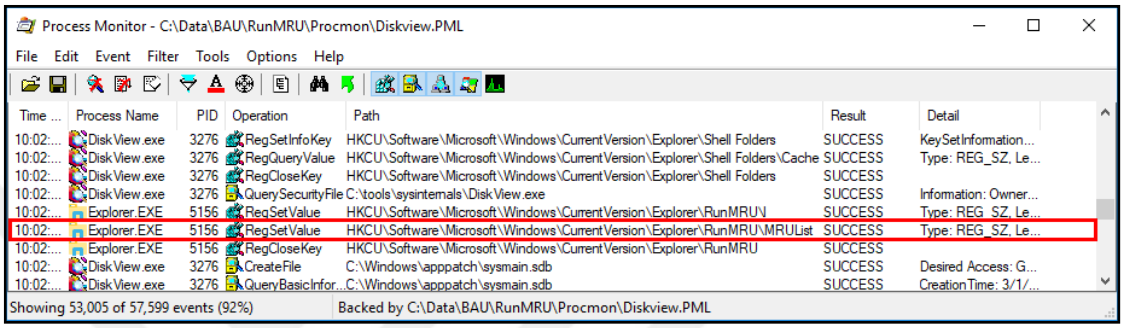
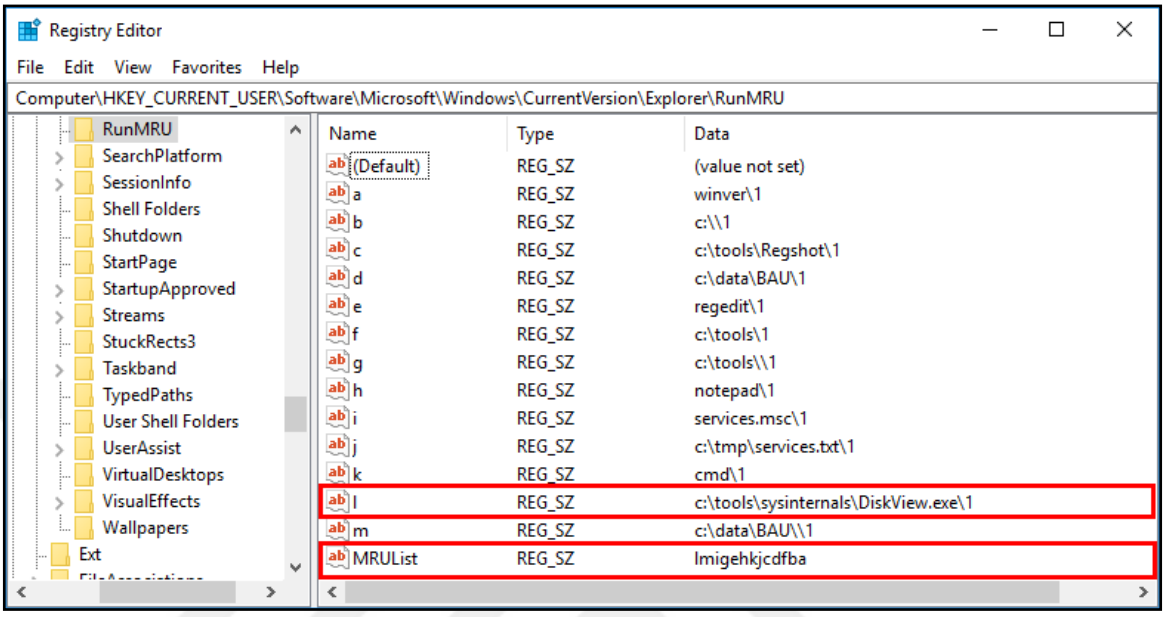


Figure 4.36 shows that "RegSetValue" event is happening after "DiskView.exe" has been executed. This means that the presence of a value under "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU" guarantees that program has been executed.

Figure 4.37: "Explorer\RunMRU" key contents



As noted earlier, "Run" commands are stored in "Most Recently Used List" structure (MSDN, 2016). Figure 4.37 shows that values under "RunMRU" key are stored in plain text format. In every "MRU List" registry structure, there is a "MRUList" registry value which stores a list of recent items' indexes. Each character corresponds to registry value name that store a "Run" command in value data. "MRUList" indexes are not order alphabetically, rather they are stored in execution order. Since English alphabet are used for indexing, maximum stored "Run" commands is 26 commands. The first character of "MRUList" corresponds to last executed "Run" command, and last character corresponds to first executed "Run" command. During this research it was found that the following algorithms is used when "Run" command is executed:

If run command already in list

Take run command index and put it first in "MRUList"

Else

Take next available English alphabet character and put it first in "MRUList" value data

Create a new value with name as alphabet character and data as the "Run" command

While "RunMRU" doesn't store date value, "MRUList" indexes order and "RunMRU" registry key last update date can both be used to determine the date of last executed "Run" command and also execution order for other "Run" commands.

Also, "RunMRU" doesn't necessary store the full path of executed "Run" command. However, other Windows system configurations information, like "Path", can be used to determine exact location of executed "Run" command.

4.3.2 Findings Summary

Analysis results of "Run" command are in Table 4.25.

Table 4.25: "Run" command findings summary

Item	"Run" command
Executable File Name	YES
Executable File Path	Partial
Executable File Size	NO
Executable File Date	NO
Executable File Hash	NO
Execution Date	Partial (Only last command)
Execution History	NO
Executions Count	NO
User	YES
Requires Reboot?	NO

4.3.3 Parsing Tools Results

Results of running different parsing tools against "Run" command are listed in Table 4.26.

Table 4.26: "Run" command parsing tools' results

Item	"Run" command
AXIOM	YES
RegRipper	YES
Registry Explorer	YES
TZWorks	NO

5. DISCUSSION and CONCLUSION

In this research, program execution traces on latest version of Windows, Windows 10 1097, were identified and studied. This research aimed to provide incident responders with the knowledge that aid them with finding and analyzing program execution artifacts even when automatic parsing tools fail to extract this information.

The main contribution of this research is analyzing and extracting information of forensics internets from six different program execution artifacts. Each artifact was checked against ten different items of forensics interests. Some of those artifacts were only poorly studied and documented before. This research also found that parsing tools often either fail to extract all possible information or fail to parse those artifacts completely.

A list of scenarios was performed to emulate real life program execution cases. Traces of programs execution were then extracted and analyzed. The results obtained were verified by comparing them against original scenarios' parameters.

Summary of six artifacts results is in Table 5.1

Table 5.1: Summary of analysis results of six program execution artifacts

Item of forensics interest	Compatibility Assistant\Store	Compatibility Assistant\Persisted	AppCompatFlags\ Layers	Appcompat Cache	bam	"Run" command
Executable File Name	YES	YES	YES	YES	YES	YES
Executable File Path	YES	YES	YES	YES	YES	Partial
Executable File Size	YES	NO	NO	NO	NO	NO
Executable File Date	NO	NO	NO	NO	NO	NO
Executable File Hash	NO	NO	NO	NO	NO	NO
Execution Date	NO	NO	NO	NO	YES	Partial (Only last command)
Execution History	NO	NO	NO	NO	NO	NO
Executions Count	YES	NO	NO	NO	NO	NO
User	YES	YES (in case saved in NTUSER.DAT)	YES (in case saved in NTUSER.DAT)	NO	YES	YES
Requires Reboot?	NO	NO	NO	YES	NO	NO

Well-known parsing tools were also used to parse those artifacts and their results were evaluated. Summary of results is in Table 5.2

Table 5.2: Parsing tools results summary of six program execution artifacts

Item	Compatibility Assistant\Store	Compatibility Assistant\Persisted	AppCompatFlags\Layers	AppCompatCache	"bam"	"Run" command
AXIOM	NO	NO	NO	NO	NO	YES
RegRipper	Partial (Only executable path)	Partial (Only executable path)	NO	YES	YES	YES
Registry Explorer	NO	NO	NO	YES	YES	YES
TZWorks	NO	NO	NO	YES	NO	NO

Results of this research should provide incident responders with better understanding of program execution artifacts, their forensics value and limitation of available parsing tools which will help them in detection stage of incident response process as it was aimed at the beginning of this research.

REFERENCES

Books

Alberts, C. & Dorofee, A., 2002. *Managing Information Security Risks: The OCTAVE (SM) Approach.* 1st ed. s.l.:Addison-Wesley Professional.

Carvey, H., 2016. *Windows Registry Forensics.* 2nd ed. s.l.:Syngress.

Periodicals

Ahmad, A., Hadgkiss, J. & Ruighaver, T., 2012. Incident response teams – Challenges in supporting the organisational security function. *Computers & Security*, **31**(5), pp. 643-652.

Friedberg, I., Skopik, F., Settanni, G. & Fiedler, R., 2015. Combating advanced persistent threats: From network event correlation to incident detection. *Computers & Security*, Volume **48**, pp. 35-57.

Mitropoulos, S., Patsos, D. & Douligeris, C., 2006. On Incident Handling and Response: A state-of-the-art approach. *Computers & Security*, **25**(5), pp. 351-370.

Other Sources

Armerding, T., 2018. *The 17 biggest data breaches of the 21st century.* [Online]
Available at: https://www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html

batcmd.com, 2017. *Background Activity Moderator Driver - Windows 10 Service.* [Online]
Available at: http://batcmd.com/windows/10/services/bam/
[Accessed 15 4 2018].

Calore, M., 2008. *A History of Microsoft Windows.* [Online]
Available at: https://www.wired.com/2008/12/wiredphotos31/

Carvey, H., 2013. *HowTo: Determine Program Execution.* [Online]
Available at: http://windowsir.blogspot.com.tr/2013/07/howto-determine-program-execution.html

Cichonski, P., Millar, T. & Scarfone, K., 2012. *Computer Security Incident Handling Guide (SP 800-61 Rev. 2),* s.l.: NIST.

Code of Federal Regulations (CRF), 2013. *CFR 164.308 - Administrative safeguards.* [Online]
Available at: https://www.law.cornell.edu/cfr/text/45/164.308

Davis, A., 2012. *Leveraging the Application Compatibility Cache in Forensic Investigations.* [Online]
Available at: https://www.fireeye.com/content/dam/fireeye-www/services/freeware/shimcache-whitepaper.pdf

Grispos, G., 2016. *On the enhancement of data quality in security incident response investigations,* s.l.: University of Glasgow.

Hameed, C., 2007. *The Program Compatibility Assistant – Part Two.* [Online]
Available at: https://blogs.technet.microsoft.com/askperf/2007/10/05/the-program-compatibility-assistant-part-two/
[Accessed 02 03 2018].

Harrell, C., 2013. *Revealing Program Compatibility Assistant HKCU AppCompatFlags Registry Keys.* [Online]

Available at: https://journeyintoir.blogspot.nl/2013/12/revealing-program-compatibility.html

IBM, 2017. *Ponemon Cost of Data Breach*, s.l.: s.n.

Ionescu, A., 2007. *Secrets of the Application Compatilibity Database (SDB) – Part 3.* [Online]

Available at: http://www.alex-ionescu.com/?p=41

ISO, 2016. *ISO/IEC 27035-1:2016 - Information technology -- Security techniques -- Information security incident management -- Part 1: Principles of incident management.* [Online]

Available at: https://www.iso.org/standard/60803.html

ISO, 2018. *ISO/IEC 27000:2018 - Information technology -- Security techniques -- Information security management systems -- Overview and vocabulary.* [Online]

Available at: https://www.iso.org/standard/73906.html

Kasperksy, 2011. *The Man Who Found Stuxnet – Sergey Ulasen in the Spotlight.* [Online]

Available at: https://eugene.kaspersky.com/2011/11/02/the-man-who-found-stuxnet-sergey-ulasen-in-the-spotlight/

Katsavounidis, C., 2018. *An Alternative to Prefetch -> BAM.* [Online]

Available at: https://www.linkedin.com/pulse/alternative-prefetch-bam-costas-katsavounidis/

Lunden, I., 2017. *After data breaches, Verizon knocks $350M off Yahoo sale, now valued at $4.48B.* [Online]

Available at: https://techcrunch.com/2017/02/21/verizon-knocks-350m-off-yahoo-sale-after-data-breaches-now-valued-at-4-48b/

Mandiant, 2017. *ShimCacheParser.* [Online]

Available at: https://github.com/mandiant/ShimCacheParser

Microsoft.com, 2012. *Understanding Shims.* [Online]

Available at: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-7/dd837644(v=ws.10)

Microsoft, 2018. *Make older programs compatible with this version of Windows.* [Online]

Available at: https://support.microsoft.com/en-us/help/15078/windows-make-older-

programs-compatible
[Accessed 03 05 2018].
MSDN, 2016. *CRecentFileList Class.* [Online]
Available at: https://docs.microsoft.com/en-us/cpp/mfc/reference/crecentfilelist-class
MSDN, 2017. *Desktop Activity Moderator.* [Online]
Available at: https://msdn.microsoft.com/en-us/windows/compatibility/desktop-activity-moderator
MSDN, n.d. *CreateService function.* [Online]
Available at: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682450(v=vs.85).aspx
[Accessed 5 4 2018].
Newman, L. H., 2016. *Hack Brief: Hackers Breach A Billion Yahoo Accounts. A Billion.* [Online]
Available at: https://www.wired.com/2016/12/yahoo-hack-billion-users/
Ragan, S., 2015. *Anthem confirms data breach, but full extent remains unknown.* [Online]
Available at: https://www.csoonline.com/article/2880352/disaster-recovery/anthem-confirms-data-breach-but-full-extent-remains-unknown.html
SANS Institute, 2017. *SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling.* [Online]
Available at: https://www.sans.org/course/hacker-techniques-exploits-incident-handling
sevenforums, 2010. [Online]
Available at: https://www.sevenforums.com/performance-maintenance/92865-q-program-compatibility-wizard.html
[Accessed 23 04 2018].
statcounter.com, 2018. *Desktop Operating System Market Share Turkey.* [Online]
Available at: http://gs.statcounter.com/os-market-share/desktop/turkey#monthly-201704-201804-bar
statcounter.com, 2018. *Desktop Operating System Market Share Worldwide.* [Online]
Available at: http://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-201704-201804-bar

statcounter.com, 2018. *Desktop Windows Version Market Share Worldwide*. [Online]
Available at: http://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide#monthly-201704-201804
Thomas, K., 2011. *PlayStation Network users reporting credit card fraud.* [Online]
Available at: https://www.csoonline.com/article/2128427/network-security/playstation-network-users-reporting-credit-card-fraud.html
Verboon, A., 2011. *Running an Application as Administrator or in Compatibility Mode.* [Online]
Available at: https://www.verboon.info/2011/03/running-an-application-as-administrator-or-in-compatibility-mode/
[Accessed 13 04 2018].