1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

01. 개요

Windows 침해사고 분석을 위해선 활성 데이터와 비활성 데이터의 수집은 필수적이다. 활성 데이터는 전원이 꺼질 경우 사라지는 휘발성 데이터로 물리 메모리, 네트워크 연결 정보, 실행중인 프로세스, 로그온 정보, 클립보드 등이 있다. 활성 데이터는 휘발 순서, 중요도에 따라 수집 순서가 정의되어 있으며 NIST가 제공하는 'Guide to Integrating Forensic Techniques into Incident Response(SP 800-86)'에서는 다음과 같은 순서로 휘발 순서를 정의하고 있다. 세션이 만료되면 정보가 사라지므로 네트워크 연결정보를 가장 먼저 수집해야 하며 이후 수집 순서는 분석가의 관점 및 사건에 따라 달라질 수 있다.

순서	데이터
1	네트워크 연결정보(Network Connections)
2	로그인 세션(Login Sessions)
3	메모리내 정보(Contents of Memory)
4	실행중인 프로세스(Running Process)
5	열려있는 파일(Open Files)
6	네트워크 구성(Network Configuration)
7	운영체제 시간(Operating System Time)

[표 1] SP 800-86 휘발성 순서(출처 : NIST, Guide to Integrating Forensic Techniques into Incident Response, SP 800-86)

비활성 데이터는 전원이 꺼진 후에도 사라지지 않는 데이터로 Metadata, 레지스트리 하이브, Event Log 등이 있다. 과거에는 디스크 이미징 후 비활성 데이터를 수집 및 분석하였지만 장치의 용량 증가로 인한 이미징 속도 저하로 라이브 포렌식 과정에서 활성 데이터 수집 시 비활성 데이터도 같이 수집하는 방식으로 변화했다. 비활성 데이터로는 [표 2]와 같은 데이터들이 존재한다.

데이터	설명
MBR	Master Boot Record의 약자로 디스크에 가장 앞부분에 기록되어 윈도우 부팅 시 필요한 필드
Metadata	파일의 위치, 수정 시간 등의 속성과 트랜잭션 로그 존재
Event Log	Windows에서 지원하는 이벤트에 대한 기록
Registry Hive	로컬 시스템과 각 계정 별 레지스트리 파일
Prefetch/Superfetch	응용 프로그램의 마지막 실행 시간, 참조 파일 목록 존재
LNK	바로가기 파일과 최근에 사용 했거나 자주 사용되는 파일 목록
Recycle.Bin	삭제된 파일들로 휴지통 내 존재하는 파일 정보
Browser Artifact	웹 브라우저의 즐겨 찾기, 방문 기록, 캐시 등
ETC	설치된 어플리케이션(웹, 메신저 등)의 로그

[표 2] 비활성 데이터 목록

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

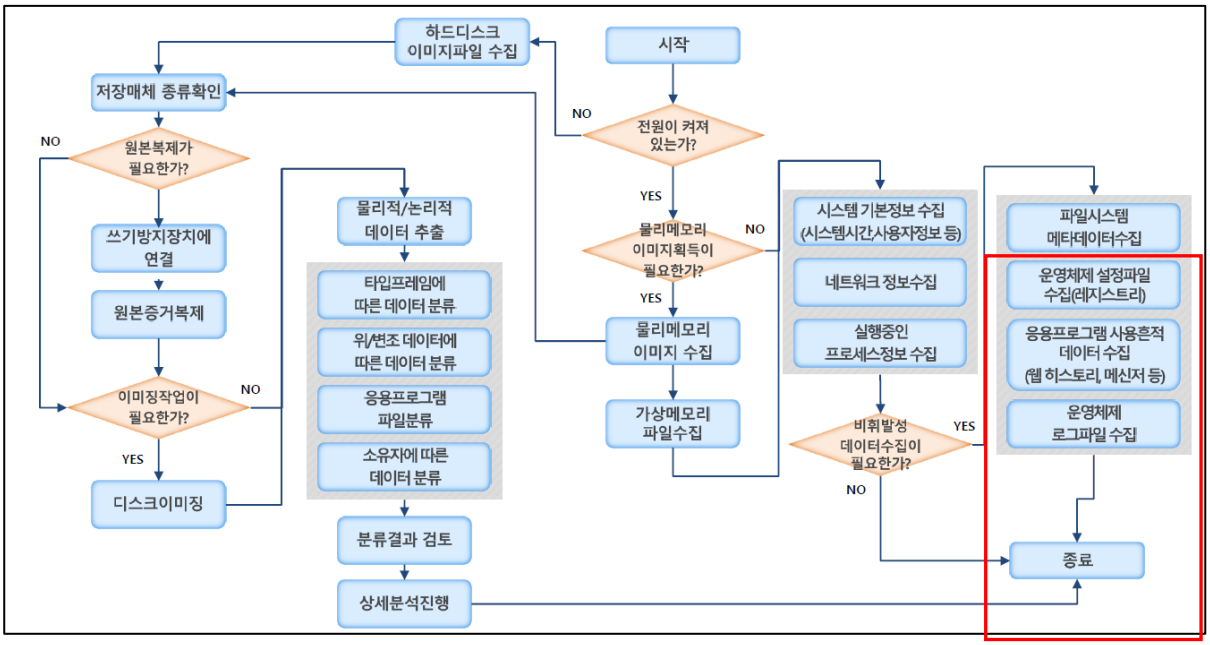
A사 침해사고 수집도구에선 위 목록에 더해 추가적인 아티팩트를 수집하여 침해사고 분석을 실시한다. [그림 1]은 침해사고 수집도구에서 수집 중인 아티팩트 중 일부이다.

No	추출 대상	상세 내용
1	Memory	Dumpit.exe, winpmem.exe를 사용하여 메모리 덤프
2	Windows IR	기존 IR 스크립트 기반으로 Ipconfig, arp, netstat 등의 정보 수집 후 txt 파일로 저장
3	Chrome	Appdata 내의 Chrome 데이터 전부를 추출 (History, Cache, Cookies 등)
4	IE	Appdata 내의 IE 데이터 추출 (WebCacheV01.dat, Index.dat)
5	Browser DATA	Chrome, IE 등의 브라우저 히스토리를 CSV 파일 형태로 추출
6	Outlook	Appdata 내의 Outlook 저장 데이터를 추출 (.ost, .pst 등)
7	Recent	최근 실행 목록
8	MetaData	$MFT, $Logfile, $UsnJrnl 수집
9	REGISTRY	레지스트리 정보 수집 (SAM, SECURITY, SOFTWARE 등)
10	USB	USB 연결기록이 저장되어 있는 setupapi.dev.log 파일 수집
11	WINEVT	윈도우 이벤트로그 수집
12	WMIC_Repository	윈도우 WMIC 설정파일
13	Autoruns	자동 실행 목록을 txt 형태로 추출 (Autoruns.exe 도구 사용)
14	ETC	Hosts, Protocol 등의 파일 추출 (system32/drivers 경로에 존재하는 파일)
15	LastActivity	마지막 파일 실행 목록을 txt 파일로 추출
16	Prefetch	응용프로그램 실행 시 생성되는 임시파일 목록
17	ThumbIconCache	윈도우 탐색기 (Explorer.exe) 상에서 출력되는 아이콘 정보를 담고 있는 파일
18	Recycle	휴지통에 존재하는 파일 리스트를 xml 파일로 추출 (파일 자체를 복사하지 않음)

[그림 1] 침해사고 수집도구 내 수집하는 아티팩트 일부

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

데이터 수집은 일반적으로 휘발성이 높은 활성 데이터 수집 후 선택적으로 비활성 데이터를 수집하므로 [그림4-2]와 같은 과정으로 데이터를 수집한다.



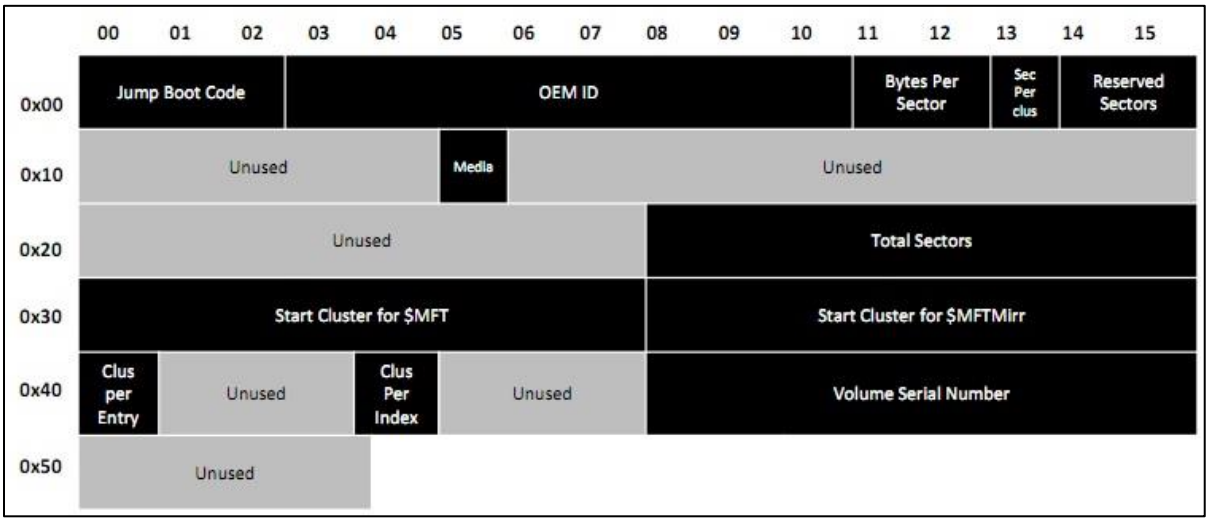
[그림 2] 침해사고분석 방법론내 아티팩스 수집 방안

[그림 2]는 증거 수집 과정 전체를 나타내고 있는데, 이번 문서에서는 빨간색 강조 처리된 비활성 데이터 수집 영역을 다룰 예정이다. 침해 발생시 악성 프로그램의 설치 및 실행 이력 확인을 위해서는 Metadata의 확인이 필요하다. Metadata란 비활성 데이터 중 파일 시스템에서 볼륨을 관리하기 위한 시스템 파일과 저장된 파일들에 대한 속성과 트랜잭션 로그를 가지고 있는 파일들의 총칭으로 NTFS 에서는 MFT(Master File Table)내에 Entry 형태로 정의 되어 있고 파일의 실제 위치 정보(Start Cluster)를 가지고 있으나 파일 참조와 같은 일반 적인 방식으로는 접근할 수 없다. 분석을 위해 Live 상태의 시스템에서 MFT에 기록된 정보를 토대로 Metadata를 수집하는 방법과 레지스트리 하이브 중 NTUSER.DAT 파일과 같이 시스템 권한으로 실행 중인 파일을 참조하는 방법에 대해서 알아보겠다.

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

02. VBR과 MFT 개요 및 아키텍처

Metadata에 접근하기 위해서는 MFT에 먼저 접근해야 한다. MFT의 위치는 고정되지 않아 볼륨의 첫 번째 클러스터에 존재하는 VBR(Volume Boot Record)을 분석해서 볼륨의 정보를 확인해야 한다. VBR은 MBR에 의해 부팅 가능한 볼륨 발견 시 해당 볼륨의 첫 번째 클러스터를 로드 시키므로 반드시 첫 번째 클러스터에 존재한다.



	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x00	EB	52	90	4E	54	57	53	20	20	20	20	00	02	08	00	00
0x10	00	00	00	00	00	F8	00	00	3F	00	FF	00	00	A8	08	00
0x20	00	00	00	00	80	00	80	00	FF	AF	50	1B	00	00	00	00
0x30	00	00	0C	00	00	00	00	00	02	00	00	00	00	00	00	00
0x40	F6	00	00	00	01	00	00	00	0B	CB	43	54	F1	43	54	5E
0x50	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	68	C0	07

[그림 3] VBR 헤더 예시 (그림 출처 : VBR 분석 (FAT32/NTFS) https://c0msherl0ck.github.io)

[그림 3]는 볼륨으로 접근해 0x00~0x5Fbytes 까지 읽은 결과이다. 색칠된 필드 이외에도 정보가 존재하지만 MFT에 접근하기 위해 필요한 데이터만 표시했다. 데이터들이 리틀 엔디안(Little Endian)으로 저장되어 있어 분석을 위해서는 빅 엔디안으로 변환하는 과정이 필요하다. 현재 파일 시스템이 NTFS를 사용하고 있는지 확인하기 위해선 0x03~0x06 필드를 확인해야 한다. OEM ID 필드며 ASCII 코드로 디코딩 시 앞 4bytes는 NTFS, 나머지 bytes는 공백문자로 채워져 있다.

다음으로 Bytes Per Sector 필드인 0x11~0x12 를 보면 0x200 이라는 값이 나오는데 1섹터의 크기가 0x200bytes(512bytes)인 것을 알 수 있다. 0x13 필드는 1클러스터당 섹터의 개수를 표현하는 필드로 현재 8개의 섹터가 1클러스터로 설정되어 있다. 마지막으로 0x30~0x37 필드는 MFT의 시작 클러스터의 위치가 표현되는 필드로 0xC0000번째 클러스터에 MFT가 존재하는 것을 알 수 있다. 정보를 종합하면 파일 시스템이 NTFS를 사용 중이고, 1클러스터의 크기가 4096bytes(0x1000)이며 MFT의 위치는 볼륨의 시작지점 부터 0xC0000000(0xC0000 * 0x1000)bytes 만큼 뒤에 존재한다는 것을 알 수 있다.

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

(1) MFT Entry

NTFS 시스템에서 Metadata는 볼륨 내 정해진 위치에 존재하지 않아 MFT Entry를 확인하여 시작 클러스터의 위치와 Data Run를 확인하여 실제 데이터에 접근해야 한다. 1개의 Entry는 2개의 섹터로 구성되어 있고 볼륨에 존재하는 모든 파일 또는 폴더는 1개 이상의 Entry를 가지게 된다. 0~15번 까지의 Entry는 예약된 영역으로 이 위치에 Metadata의 Entry가 존재하게 된다.

Index	파일명	설명
0	$MFT	모든 MFT Entry를 가지고 있는 파일
1	$MFTMirr	MFT 0~3번 Entry의 백업 파일
2	$LogFile	메타데이터의 트랜잭션이 저장된 파일
3	$Volume	볼륨 레이블, 식별자, 버전이 저장된 파일
4	$AttrDef	속성의 식별자, 이름, 크기가 저장된 파일
5	.	볼륨의 루트 디렉터리(ex. C:\)
6	$Btimap	볼륨의 클러스터 할당 정보가 저장된 파일
7	$Boot	부트 레코드의 정보가 저장된 파일
8	$BadClus	배드 섹터를 가진 클러스터의 정보가 저장된 파일
9	$Secure	파일의 보안과 접근 제어 정보가 저장된 파일
10	$Upcase	모든 유니코드 문자의 대문자가 저장된 파일
11	$Extend	확장 파일($ObjID, $Quota, $Reparse, $UsnJrnl 등)의 정보 저장
12~15	미사용	미래를 위해 예약되어 있는 Entry
16~23	미사용	비어있는 영역(Sparse)
24~	일반 파일	일반 파일이 기록되기 시작하는 Entry

[표 3] MFT Entry 예약 영역

침해사고 분석 시 사용자 행위 분석을 위해 주로 수집하는 파일은 3가지로 $MFT, $LogFile, $UsnJrnl 이 있다. $MFT 파일은 모든 파일의 생성, 접근, 수정 시간과 소유자 정보를 가지고 있다. 하지만 MFT 파일은 마지막 작업 시간만 기록되기 때문에 상세한 작업 기록 타임 라인을 확인 하기 위해선 $LogFile과 $UsnJrnl 파일을 확인할 필요가 있다.$LogFile은 파일 및 폴더 생성, 삭제, 내용 변경, MFT Entry 내용 변경과 같은 트랜잭션 작업 내용을 기록하여 트랜잭션 오류 발생 시 복구를 위한 파일이다. 정전이나 기타 오류 발생 시 진행 중이던 작업을 이전 상태로 복구하며 일반적으로 64MB가 할당되어 있다.

$UsnJrnl 파일은 $Extend 폴더 내에 존재하며 MFT 예약 영역에 포함되어 있지 않다. 기록되는 내용은 LogFile과 비슷하나 오류 복구 목적이 아닌 트랜잭션 작업이 끝난 후의 내용을 기록한다. $UsnJrnl은 $Max 와 $J 두 개의 속성이 존재 하는데, $Max는 저널 데이터의 최대 크기, 할당 크기가 저장되어 있고 32bytes의 고정 크기를 가진다. $J는 저널 데이터가 Entry 형태로 저장되는 파일로 시간 순으로 저장된다. 새로운 데이터는 $J 속성 끝에 저장되며 허용된 크기를 초과할 경우 앞부분을 0으로 채워 Sparse 영역으로 만든다. Sparse 영역을 포함하여 수집할 경우 용량이 과도하게 측정될 수 있어 실제 데이터가 저장된 영역만 수집하는 것이 자원 측면에서 유리하다.

(2) $DATA 속성

Entry 내부에는 여러가지 속성이 있지만 Metadata 수집을 위해서는 $DATA 속성이 가장 중요하다.[그림 4]는 MFT 0번 Entry의 $DATA 속성이다.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x00	80	00	00	00	68	00	00	00	01	00	40	00	00	00	06	00
0x10	00	00	00	00	00	00	00	00	3F	C9	03	00	00	00	00	00
0x20	40	00	00	00	00	00	00	00	00	00	94	3C	00	00	00	00
0x30	00	00	94	3C	00	00	00	00	00	00	94	3C	00	00	00	00
0x40	33	20	C8	00	00	00	0C	33	0C	C8	00	F4	70	37	33	07
0x50	C8	00	25	7A	73	43	A7	22	01	15	26	C7	00	32	66	4E
0x60	8A	7B	AE	00	00	00	00	00	B0	00	00	00	48	00	00	00

[그림 4] 0번 Entry의 $DATA 속성

0x00~0x03 영역은 $DATA 속성을 알려주는 영역으로 속성의 시그니쳐는 80으로 시작한다. 파일의 데이터가 저장되는 영역으로 700바이트보다 작을 경우 거주 속성이 되어 현재 Entry 내부에 데이터가 저장되게 된다.

0x08~0x08 영역은 거주 속성과 비거주 속성을 구분할 수 있는 플래그로 1일 경우 비거주 속성이다. 0x00~0x15 영역은 Common Attribute Header로 거주 속성 여부에 상관없이 공통적으로 존재한다.

마지막으로 0x20~0x21 영역은 데이터 런이 시작되는 위치를 표시하는 값으로 정수 형태로 변환 시 64이다. 따라서 $DATA 속성의 시작 위치를 기준으로 64바이트 만큼 떨어진 위치에 데이터 런이 존재하는 것을 알 수 있다.

거주 속성은 시작 위치부터 24바이트 떨어진 위치부터 속성 내용이 존재하게 된다. 일반적으로 $DATA 속성이 차지하는 공간이 크기 때문에 데이터의 내용이 700바이트 이하라면 거주 속성으로 존재하게 되며 비거주 속성으로 지정되게 되면 데이터의 크기만큼 클러스터를 할당 받게 된다.

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

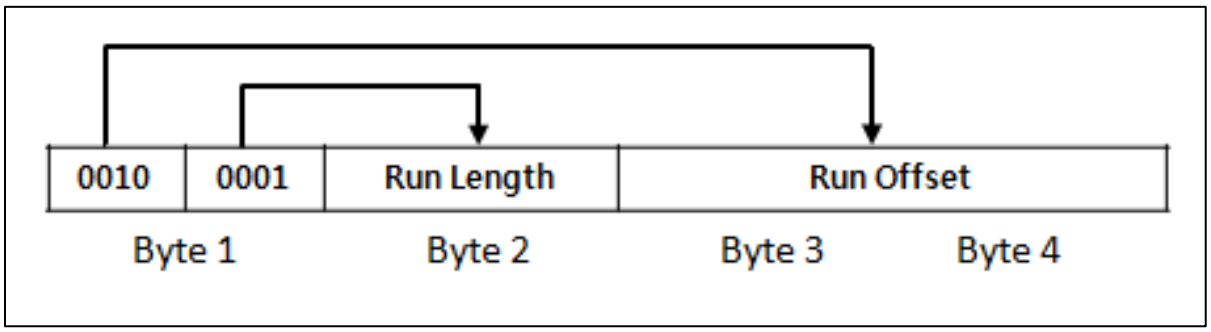
(3) 데이터 런

수집 할 파일의 속성이 비거주 속성일 경우 데이터가 실제로 저장된 클러스터를 찾아야 한다. 최소 한 개의 클러스터가 할당되며 저장된 영역이 연속적이지 않을 수 있다.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0x00	80	00	00	00	68	00	00	00	01	00	40	00	00	00	06	00
0x10	00	00	00	00	00	00	00	00	3F	C9	03	00	00	00	00	00
0x20	40	00	00	00	00	00	00	00	00	00	94	3C	00	00	00	00
0x30	00	00	94	3C	00	00	00	00	00	00	94	3C	00	00	00	00
0x40	33	20	C8	00	00	00	0C	33	0C	C8	00	F4	70	37	33	07
0x50	C8	00	25	7A	73	43	A7	22	01	15	26	C7	00	32	66	4E
0x60	8A	7B	AE	00	00	00	00	00	B0	00	00	00	48	00	00	00

[그림 5] 0번 Entry의 런리스트

영역은 $DATA 속성을 알려주는 영역이다. 파일의 데이터가 저장되는 영역으로 700바이트보다 작을 경우 거주 속성이 되어 현재 Entry 내부에 데이터가 저장된다. 반대의 경우 [그림4-6]와 같이 클러스터의 오프셋과 길이가 표현된다.

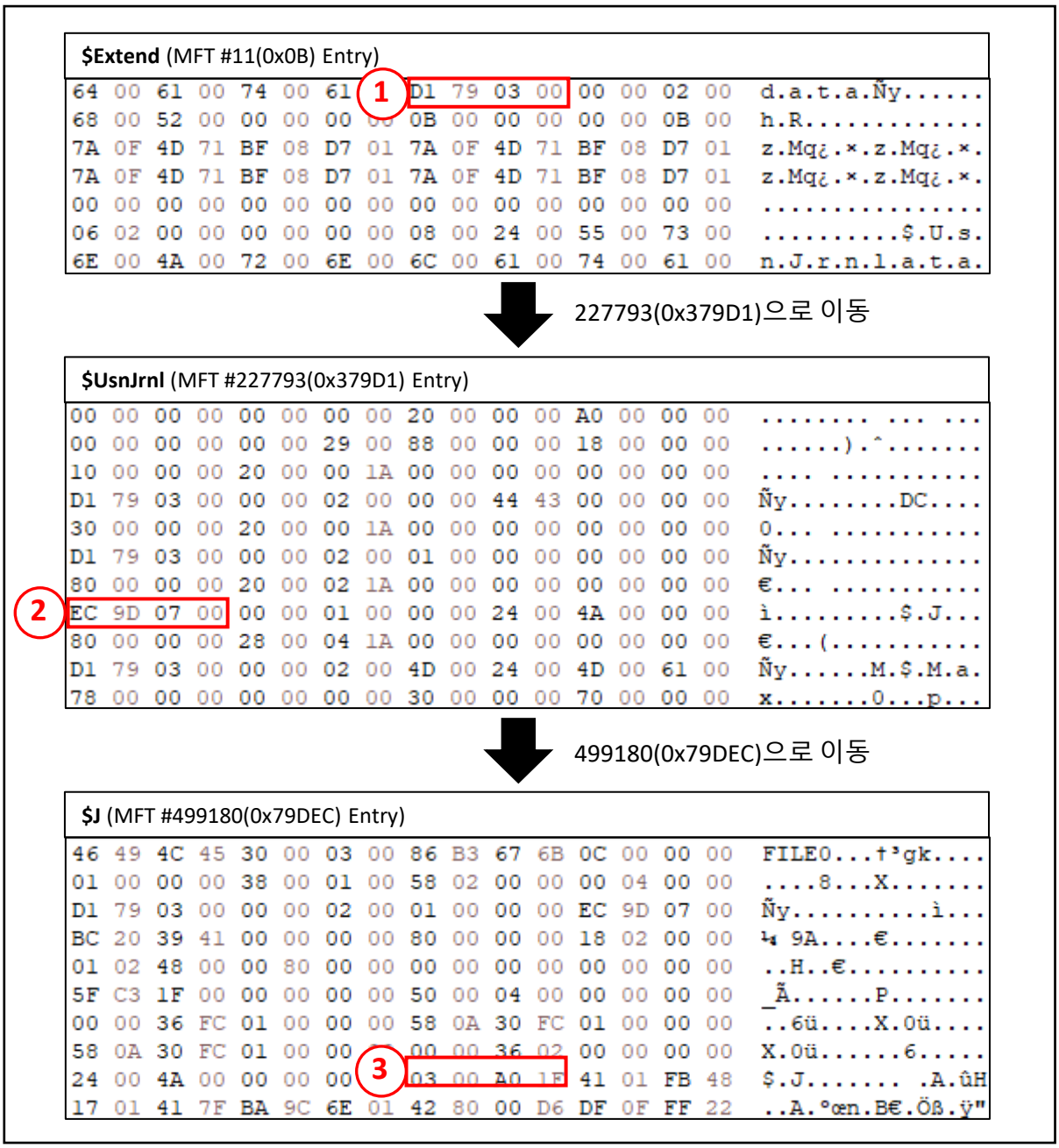


[그림 6] 데이터 런 구조(출처 : FORENSIC-PROOF, NTFS – 속성 (Attributes))

파란색으로 표시된 바이트는 데이터 런의 헤더로 앞 4비트는 클러스터의 오프셋을, 뒤 4비트는 클러스터의 길이를 나타낸다. 0x55~0x5C 데이터 런을 보면 클러스터의 오프셋은 4바이트, 클러스터의 길이는 3바이트이다. 클러스터의 길이는 헤더 바로 뒤에 나오는 3바이트를 빅 엔디안으로 변환하면 0x0122A7이 된다. 클러스터의 오프셋은 길이 뒤에 나오는 4바이트로 0x00C72615 이다. 첫 번째 데이터 런의 오프셋은 LCN(Logical Cluster Number)으로 볼륨의 첫 번째 클러스터부터 떨어진 길이이다. 두 번째 부터의 데이터 런은 파일의 첫 번째 클러스터 부터 떨어진 길이이다. 만약 클러스터의 오프셋이 지정되지 않고 길이만 지정되어 있는 경우 해당 영역은 Sparse로 모든 바이트가 00으로 채워져 있다.

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

$MFT와 $LogFile은 예약된 Entry에 정적으로 존재해서 수집하기 쉽지만 $UsnJrnl:$J 파일은 $Extend 폴더 하위에 존재해서 경로를 탐색할 필요가 있다. 또한 $J는 $UsnJrnl 파일의 ADS(Alternate Data Stream)로 $ATTRIBUTE_LIST 속성을 참조해야 한다.



[그림 7] $UsnJrnl:$J 탐색 과정

[그림 7]의 ①번 값은 $MFT 11번 Entry의 $UsnJrnl이 존재하는 영역으로 $INDEX_ROOT 하위 노드의 헤더 0x00~0x03에 노드의 Entry 번호를 나타낸 것으로 0x379D1Entry에 $UsnJrnl Entry가 존재하는 것을 알 수 있다. ②번 값은 $UsnJrnl의 $ATTRIBUTE_LIST 노드 중 $J의 Entry 번호로, 현재 $J가 0x79DEC Entry에 존재하고 있다. 마지막으로 ③번 값은 $J Entry에 접근해 데이터 런이 시작되는 영역을 나타낸 것으로 현재 클러스터 0x1FA000개가 오프셋이 지정되지 않았는데, 이는 Sparse 처리된 영역으로 다음 데이터 런부터 계산하면 실제 데이터를 수집할 수 있다.

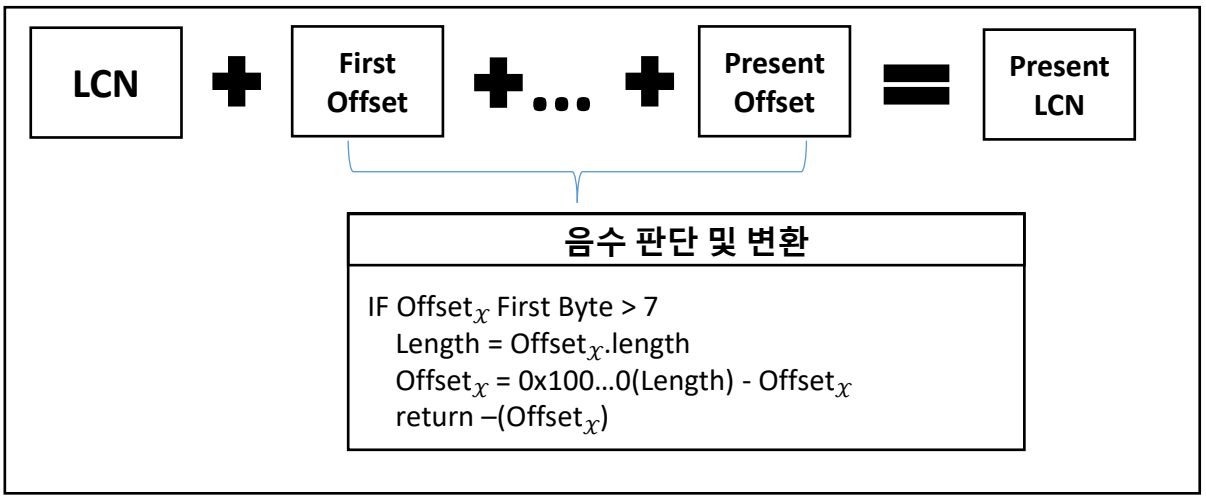
1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

데이터를 수집하기 위해선 클러스터의 상대적인 오프셋이 아닌 첫 번째 클러스터의 LCN을 바탕으로 계산하는 것이 편리하다. 클러스터의 오프셋을 계산하기 위해선 2의 보수에 대한 이해가 필요하다. 본 문서에서는 클러스터의 오프셋 연산에 집중하기 위해 2의 보수 방법에 대해서는 생략하고자 한다.

순서	데이터 런	길이	오프셋	2의 보수
1	03 00 B0 1F	0x1FB000	-	-
2	41 60 61 FF D2 00	0x60	0x00D2FF61	-
3	32 80 00 EA 60 B4	0x80	0xB460EA	0x4B9F16
4	22 80 00 45 FF	0x80	0xFF45	0xBB
5	42 80 00 F9 83 C3 01	0x80	0x01C383F9	-

[표 3] 데이터 런 계산 과정

[표 3]는 $UsnJrnl:$J의 런 리스트 일부이다. 첫 번째 데이터 런은 길이만 존재하고 오프셋은 존재하지 않는다. 실제 데이터가 존재하지 않는 영역으로 Sparse 영역이다. 두 번째 데이터 런의 오프셋은 첫 번째 데이터 런이 Sparse 였으므로 LCN이 된다. 세 번째 데이터 런의 오프셋은 음수이므로 2의 보수로 변환할 경우 0x4B9F16이 되며, LCN을 구하기 위한 계산식은 (0x00D2FF61) + (-0x4B9F16) = 0x87604B 이다.

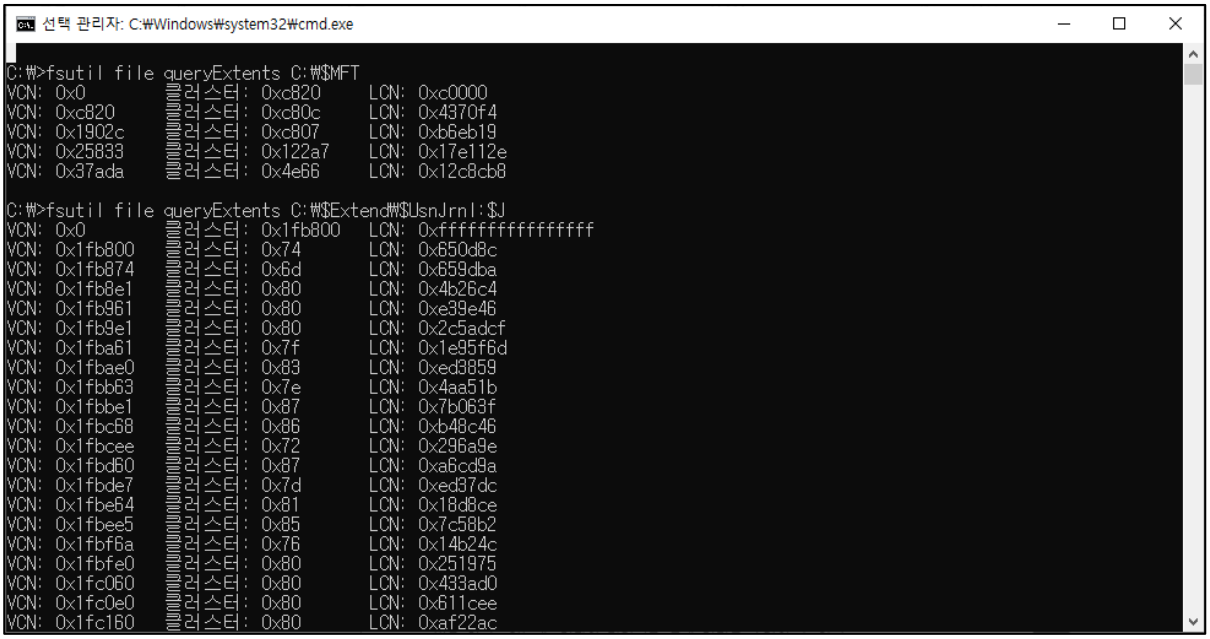


[그림 8] 데이터 런 계산 수식

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

정리하자면 데이터 런의 오프셋의 첫 번째 비트가 0(양수)이면 합 연산을 통해 LCN을 구할 수 있고, 음수일 경우 2의 보수로 변환하여 LCN과의 차를 구해야 한다.

MFT의 위치를 찾는 것 부터 데이터 런을 계산하여 실제 파일이 저장된 클러스터의 위치를 찾는 방법을 알아보았는데, 간단한 명령어를 통해 결과를 출력할 수 있다. Windows Server 2008, Windows 7 이후 부터 지원되는 fsutil의 queryExtents 명령을 사용하면 [그림4-9]와 같이 클러스터 단위의 LCN과 길이가 출력된다.



[그림 9] fsutil 명령 사용 결과

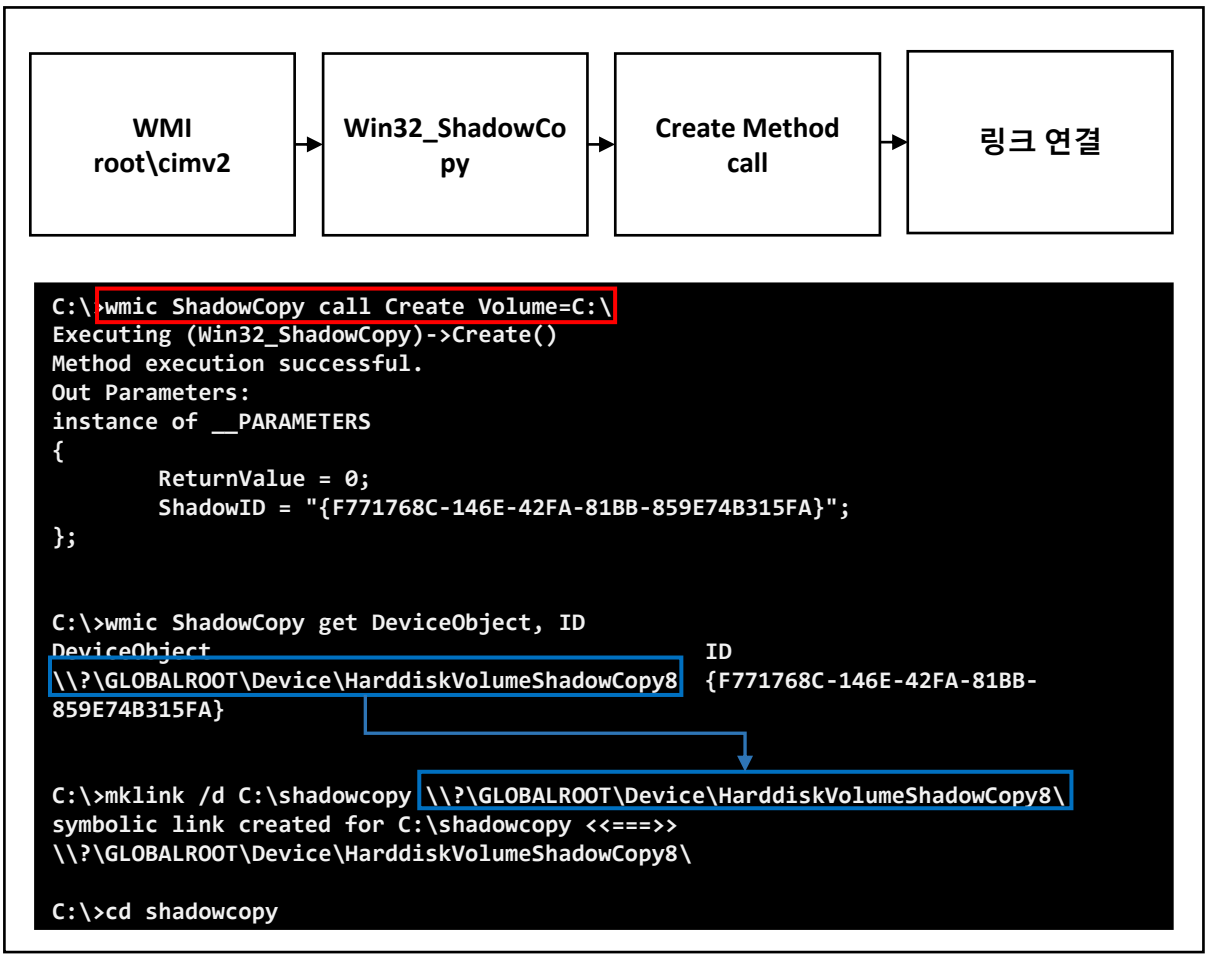
1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

03. 시스템 점유 파일 접근

(1) 시스템 점유파일 접근 매커니즘 설명

NTUSER.DAT, UsrClass.DAT, SAM 등의 레지스트리 하이브 형태의 비활성 데이터는 온라인 상태에서 수집을 시도할 경우 System 권한으로 실행 중으로 작업이 실패했다는 메시지가 출력된다. 위 파일들도 물론 데이터 런을 분석하여 수집할 수 있지만 다른 방법을 소개해보겠다.

Vista 이후의 NTFS 에서는 백업 및 복원 시 VSS(Volume Shadow Service)를 사용해 복원 지점에 대한 스냅샷을 생성한다. Volume Shadow 복사본 생성을 요청할 경우 쓰기에 대한 I/O 작업이 중단되며 기본적으로 정의된 기록기(Writers)의 수집 파일 목록을 참조하여 복사본을 만들게 된다. 수집 대상이 레지스트리 하이브는 Registry Writer 라는 기록기에 의해 복사본이 만들어지게 되는데, HKLM₩System₩CurrentControlSet₩Control₩hivelist 하위 값에 모든 사용자에 대한 하이브 위치가 정의되어 있다. 앞에서 레지스트리 수집 시 온라인 상태에선 접근이 제한된다고 설명했는데, Volume Shadow로 생성된 복사본은 오프라인 상태이다. 또한, 실제 볼륨에 접근하는 방식과 똑같이 복사본에 접근할 수 있어 탐색 및 수집이 간단하다.



[그림 10] Volume Shadow 생성 및 링크 연결

Volume Shadow 생성 시 용량 효율을 위해 hiberfil.sys, pagefile.sys 파일 등은 제외되지만 Windows 10 기준 복사본 한 개당 약 1GB 정도의 공간을 차지하게 된다. 시스템 설정 변경을 통해 주기적으로 복원 지점을 생성할 수 있고, 복원을 위한 최대 크기를 조절할 수 있다.

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

(2) 소스코드를 통한 시스템 점유파일 접근 및 메타데이터 접근 방안

```
class VSS:
    def hiveList(self):
        try:
            roothandle = winreg.ConnectRegistry(None, winreg.HKEY_LOCAL_MACHINE)
            key = winreg.OpenKey(roothandle,
"SYSTEM\\CurrentControlSet\\Control\\hivelist", 0,
            winreg.KEY_READ | winreg.KEY_WOW64_64KEY)
            info = winreg.QueryInfoKey(key)

        except Exception as e:
            return []

        resultPath = list()
        resultName = list()
        for i in range(info[1]):
            try:
                searchReg = winreg.EnumValue(key, i)[1]
                resultName.append(winreg.EnumValue(key, i)[0])
                convertPath = searchReg.split("\\")[3:]
                convertPath = '\\'.join(convertPath)
                resultPath.append(convertPath)
            except Exception as e:
                continue
        try:
            winreg.CloseKey(key)
        except Exception as e:
            pass
        return resultPath, resultName

    def vssList(self):
        wcd=win32com.client.Dispatch("WbemScripting.SWbemLocator")
        wmi=wcd.ConnectServer(".","root\cimv2")
        obj=wmi.ExecQuery("SELECT * FROM win32_ShadowCopy")
        return [x.DeviceObject for x in obj]

    def vssCreate(self, drive):
        wmi=win32com.client.GetObject("winmgmts:\\\\.\\root\cimv2:Win32_ShadowCopy")
        createmethod = wmi.Methods_("Create")
        createparams = createmethod.InParameters
        createparams.Properties_[1].value="{0}\\".format(drive)
        results = wmi.ExecMethod_("Create",createparams)
        return results.Properties_[1].value

class Metadata:
    def int32(self, x):
        pad = "0x"
        count = 1
        while True:
            if len(pad) == len(hex(x)):
                break
            pad += "F"
            count += 1
        con_hex = int(pad, 16)
        return -(con_hex+1 - x)
```

```
   def open_windows_partition(self,letter,mode="rb", buffering=-1, encoding=None,
errors=None,
        newline=None, closefd=True, opener=None):
        return open(fr"\\.\{letter}:", mode, buffering, encoding, errors, newline,
closefd, opener)

    def __vbr_structure(self):
        self.drive.seek(0)
        vbrSector = self.drive.read(512)
        ntfsCheck = vbrSector[3:7].decode('ascii')
        if ntfsCheck != "NTFS":
            print("[FAIL] this is not a NTFS system.")
            quit()
        bps = struct.unpack('<L', vbrSector[11:13] + b"\x00\x00")[0]
        spc = struct.unpack('<L', vbrSector[13:14] + b"\x00\x00\x00")[0]
        csize = bps * spc
        moffset = struct.unpack('<L', vbrSector[48:52])[0] * csize
        self.structure = {
            "BPS": bps,
            "SPC": spc,
            "ClusterSize": csize,
            "MFToffset": moffset
        }

    def queryExtents(self):
        dataRuns = os.popen("fsutil file queryExtents \"{0}\"
csv".format(self.filename)).read()
        length = list()
        offset = list()
        for runlist in dataRuns.split("\n"):
            if runlist == "" or runlist.find("VCN") != -1:
                continue
            length.append(self.convert_byte(runlist.split(",")[1]))
            offset.append(self.convert_byte(runlist.split(",")[2]))
        return length, offset

    def cluster_parse(self):
        with open(wFileName, "wb") as f:
            for i in range(idx):
                if hex(offset[i]).upper().find("0XFFFFFF") != -1:
                    print("[{0}] Sparse Area  Length : {1}".format(i,
hex(self.cluster_size(length[i]))))
                    continue
                else:
                    self.drive.seek(0)
                    self.drive.seek(self.cluster_size(offset[i]))
                    f.write(self.drive.read(self.cluster_size(length[i])))
                    fileSize += self.cluster_size(length[i])
```

[그림 11] 레지스트리 하이브 및 메타데이터 수집 소스코드 일부

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

Volume Shadow를 활용해서 레지스트리 하이브를 수집하는 방법과 MFT Entry를 분석해 Metadata를 수집하는 소스코드 중 일부이다. VSS 클래스에서는 크게 세가지 기능이 필요하다. 첫 번째 기능은 hiveList(line 2~27) 함수에서 수행하고 있다. 수집을 위해 하이브가 존재하는 경로를 알아야 하기 때문에 레지스트리 경로SYSTEM₩CurrentControlSet₩Control₩hivelist의 하위 값들을 참조 하여 하이브 이름과 경로를 반환한다.

두 번째 기능은 현재 상태의 Volume Shadow를 생성하는 것이며 vssCreate(line 29~33) 함수에서 수행하고 있다. WMI root₩cimv2 네임스페이스의 Win32_ShadowCopy 클래스에서 현재 볼륨에 대한 Create Method를 요청하여 생성한다. 세 번째 기능은 방금 전 생성한 Volume Shadow를 특정하는 기능이다. Win32_ShadowCopy에 질의하여 모든 개체를 반환하며 vssList(line 29~33) 함수에서 수행된다. 모든 과정이 완료된 후 ₩₩?₩GLOBALROOT₩Device₩HarddiskVolumeShadowCopy1₩ + 접근할 파일 경로(ex. Windows₩System32₩config₩SYSTEM) 형태로 파일을 핸들링 하게 된다.

```
C:\igloo>python technote.py -R
[DONE] "SYSTEM" copy done.. (C:\igloo\RegHive_220624105124\SYSTEM)
[DONE] "SOFTWARE" copy done.. (C:\igloo\RegHive_220624105124\SOFTWARE)
[DONE] "DEFAULT" copy done.. (C:\igloo\RegHive_220624105124\DEFAULT)
[DONE] "SECURITY" copy done.. (C:\igloo\RegHive_220624105124\SECURITY)
[DONE] "SAM" copy done.. (C:\igloo\RegHive_220624105124\SAM)
...
```

[그림 12] 레지스트리 하이브 수집 실행 결과 예시

Metadata 클래스는 데이터의 Literal를 통일하는 것과 비트 연산 기능의 구현이 가장 중요하다. 클래스가 실행되면 우선 현재 볼륨에 대한 핸들링을 위해 open_windows_partition(line 56~58) 함수가 수행한다. 그 후 VBR에서 클러스터의 크기, $MFT의 위치들을 수집하며 __vbr_structure(line 60~76)에서 수행한다. 엔디안을 변환하기 위해 struct.unpack('<L', DATA) 함수를 사용했는데, 기본적으로 4Bytes에 대한 결과만 제공되어 데이터에 Null 값의 패딩이 추가됐다.

1 행위 분석을 위한 구조 분석 및 추출 방안 : Data Run 분석을 통한 Cluster 추출

다음으로 수집 대상 파일의 데이터 런을 파싱하게 되며 queryExtents(line 78~87)에서 수행되고 시스템 명령어(fsutil file queryExtents FILENAME)를 사용해 오프셋과 길이가 정수형으로 변환 후 반환된다. queryExtents 명령어가 사용 불가능할 경우 MFT 예약 Entry내의 파일로 판단해 $DATA의 데이터 런을 수집하게 된다. 이 과정에서 2의 보수로 변환하는 함수가 int32(line 45~54)이다. 변환할 16진수의 길이로 자리 올림수를 동적으로 계산하여 두 수간의 차로 2의 보수로 변환한 결과를 반환하게 된다. 결과적으로 cluster_parse(line 89~99) 함수에서 데이터 런에 따라 파일의 복사가 수행된다.

```
C:\igloo>python technote.py -M C:\$Extend\$UsnJrnl:$J
{'BPS': 512, 'SPC': 8, 'ClusterSize': 4096, 'MFToffset': 3221225472}
[0] Sparse Area  Length : 0x201000000
[1] Start Offset : 0xf9593a000  End Offset : 0xf9594f000  Length : 0x15000
[2] Start Offset : 0xeec154000  End Offset : 0xeec1cf000  Length : 0x7b000
[3] Start Offset : 0x12412e9000  End Offset : 0x124136e000  Length : 0x85000
[4] Start Offset : 0x11eef94000  End Offset : 0x11ef00f000  Length : 0x7b000
[5] Start Offset : 0x488f8c000  End Offset : 0x48900c000  Length : 0x80000
[6] Start Offset : 0xfa37b6000  End Offset : 0xfa383c000  Length : 0x86000
[7] Start Offset : 0x11f0c1f000  End Offset : 0x11f0c99000  Length : 0x7a000
[8] Start Offset : 0x10608c6000  End Offset : 0x106094e000  Length : 0x88000
    ...
[69] Start Offset : 0xaadea8000  End Offset : 0xaadf28000  Length : 0x80000
[70] Start Offset : 0x126c8ed000  End Offset : 0x126c96d000  Length : 0x80000
[DONE] File Size : 34144256 bytes
```

[그림 13] Metadata 수집 실행 결과 예시

결론을 살펴보면 해당 분석결과는 오프셋에 따라 커서가 이동하여 해당하는 길이만큼 복사하기 때문에, 추가적으로 Sparse 영역을 구분할수 있어서 불필요한 공간 낭비를 방지할 수 있다는 점을 발견할 수 있었다. (기존 상용 소프트웨어 대비 절반이상 공간 낭비를 방지 가능한 것으로 확인)

05. 결론

앞서 MFT를 이용한 Metadata 수집과 Volume Shadow Copy를 이용한 레지스트리 하이브 수집 방법에 대해 알아보았다. kernel32를 이용한 파일 핸들링으로 시스템 점유 파일에 접근하는 등 데이터를 수집하는 방법은 다양하기 때문에 수집 대상 환경과 침해 유형을 파악하여 적절한 도구를 사용해야 한다. 행위 분석을 위해서는 활성 데이터 뿐만 아니라 비활성 데이터의 수집이 필수불가결이 되었다. 물론 데이터 수집을 보다 쉽게 획득할 수 있는 도구들이 다양하지만, 도구에서 활용되는 동작 원리를 상세하게 설명한 자료들은 부재하다고 볼 수 있다. 본 문서를 통해서 수집 도구의 동작 구조를 이해하고 파일 시스템이 데이터를 저장하는 과정에 대한 상세한 내용을 숙지할 수 있는 기회가 될 수 있기를 기대한다.

06. 참고 자료

[1] NIST SP 800-86
https://csrc.nist.gov/publications/detail/sp/800-86/final
[2] VBR 분석 (FAT32/NTFS)
https://c0msherl0ck.github.io/file%20system/post-VBR/
[3] Concept - Data Runs
http://inform.pucp.edu.pe/~inf232/Ntfs/ntfs_doc_v0.5/concepts/data_runs.html
[4] NTFS Documentation by Richard Russon and Yuval Fledel
https://dubeyko.com/development/FileSystems/NTFS/ntfsdoc.pdf
[5] 김동건, 박석현 and 조오현. (2020). 삭제된 $UsnJrnl 파일 복구를 통한 과거 사용자 행위 확인. 융합정보논문지, 10(5), 23-29.
https://www.kci.go.kr/kciportal/ci/sereArticleSearch/ciSereArtiView.kci?sereArticleSearchBean.artiId=ART002588372
[6] NTFS – 속성 (Attributes)
http://forensic-proof.com/archives/590
[7] Volume Shadow Copy Service
https://docs.microsoft.com/en-us/windows-server/storage/file-server/volume-shadow-copy-service
[8] Registry Backup and Restore Operations Under VSS
https://docs.microsoft.com/en-us/windows/win32/vss/registry-backup-and-restore-operations-under-vss