



작성자	김서영, 윤지원
분석 일자	2024.05.26.~2024.06.09.
작성 일자	2024.05.26.~2024.06.09.
분석 대상	USB-002.dd
문서 버전	1.0
작성자 E-mail	sykim1802@naver.com

0. 목차

1. 문제3

2. 분석 도구3

3. 환경3


4. Write-Up.....4

5. Flag..... 13

6. 별도 첨부 14

7. Reference 15

1. 문제

URL	-
문제 내용	The new employee A asked the superior B for the access password of the old file server. B said, "It's a complicated password so I didn't memorize it. So, I saved the password in USB". They found the USB was in a warehouse, but it didn't mount. B said, "Maybe the USB is encrypted, but I don't remember the password. Repair the USB and access the file server!".
문제 파일	<div>  <div>401 - Irresponsible predic</div> </div>
문제 유형	Disk forensics
난이도	3 / 3

2. 분석 도구

도구명	다운로드 링크	Version
010editor	SweetScape Software Inc - Download 010 Editor	14.0.1

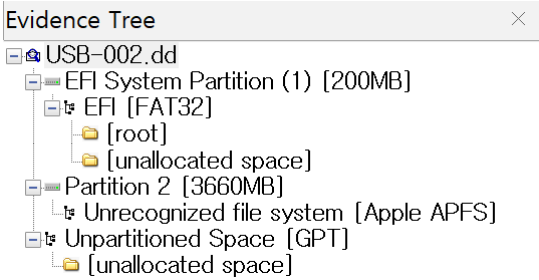
3. 환경

OS
Windows11 x64

4. Write-Up

파일명	USB-002.dd
용량	3.77GB
SHA256	32970ecf2384db786b2ed832ef2f62af465b37baab3edf4bcd0473d5dc3dcff8
Timestamp	2024-05-26 02:46:26

1. Repair the USB and then find password hint for the USB. (200 points)

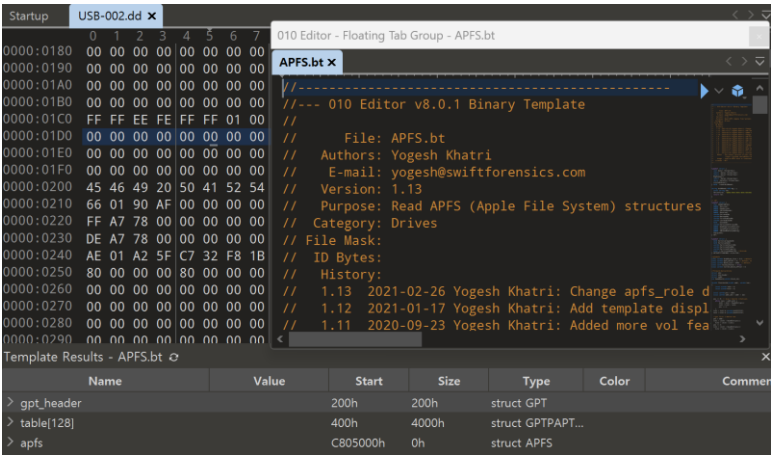


[사진 1] FTK Imager로 열어본 USB-002.dd

FTK Imager 로 USB-002.dd 를 열어보았다.

USB-002.dd 는 GPT 구조를 가지고 있고, EFI Partition1 과 APFS 구조의 파티션 2 를 가지고 있다는 것을 확인했다. 인식되지 않는 부분이 너무 많아 010editor 로 더 살펴보기로 했다.

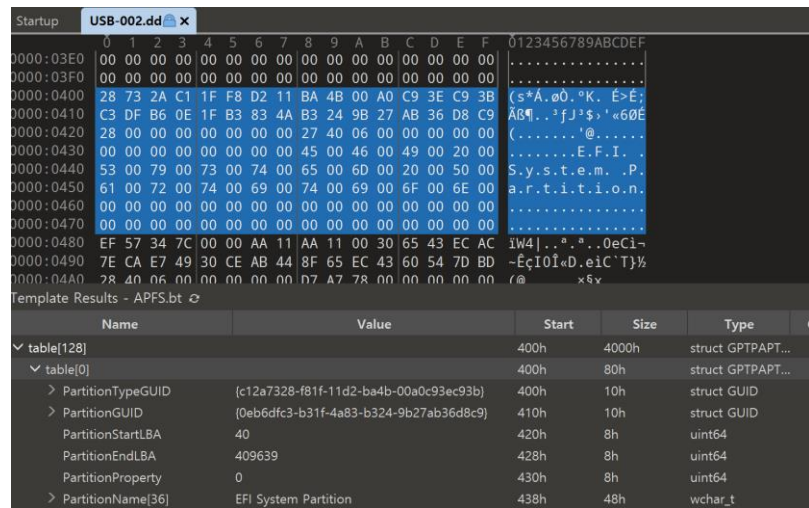
010editor 로 연 후, 010editor 의 template 중 APFS.bt template 이 GPT 구조와 APFS 구조에 대한 정의와 식별을 담고 있다는 것을 찾았고, USB-002.dd 에 실행시켰다.



[사진 2] APFS.bt template 실행 결과창

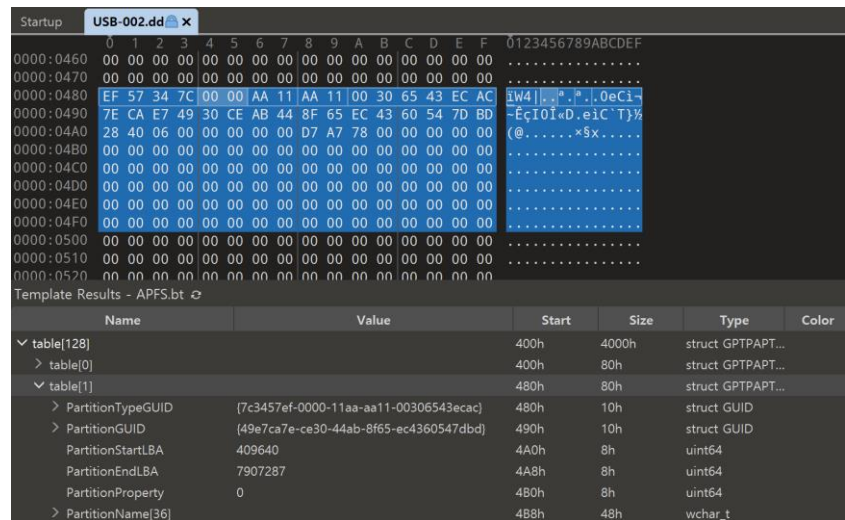
Template 결과창으로 gpt_header, table[128], apfs 구조의 블록 하나가 식별되었다.

우선 table 들을 열어보았다.



[사진 3] EFI Partition

사진 1 에서 확인한 첫 번째 파티션인 EFI System Partition 이 table[0]이라고 추정했다.



[사진 4] table[1]

그 다음 파티션인 Partition2 가 table[1]일 것으로 추측했다.

Apple APFS container
APFS FileVault volume container 7C3457EF-0000-11AA-AA11-00306543ECAC

[사진 5] table[1]의 Partition Type GUID 검색 결과

Partition Type Guid List([\(List of GPT partition table GUIDs \(toomanyatoms.com\)\)](http://List of GPT partition table GUIDs (toomanyatoms.com)))에서 table[1]의 값을 검색하여 APFS 를 사용하는 파티션임을 확인했다.

StartLBA 와 EndLBA 또한 나와있어 USB-002.dd 의 Partition2 의 크기 및 구조를 파악했다.

[WHS-2] .iso

[사진 4]에서 확인할 수 있는 PartitionStartLBA hex 값: 0x64028 (리틀레디안 방식으로 읽음)

Startup	USB-002-recover.dd x															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0C80:4FF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:50A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

[사진 6] 0x64028 위치

APFS 구조에서 Main Superblock 이 시작되어야 하는 위치인데 00 으로 뒤덮여있다.

이 USB 를 repair 하는 것이 심상치 않음을 느꼈다.

한글로 APFS 구조에 대해 자세히 설명되어 있는 글이 거의 없어서 영어 논문과 영어 사이트들을 찾아보며 APFS 구조에 대해 공부했다. Reference 에 첨부해 두었다.

APFS 구조에서는 Superblock 의 복사본이 다른 블록들의 파티션에 저장되어 있다. Main SuperBlock 을 위해 Checkpoint Superblock 을 사용해도 된다는 뜻이다. 또한, 가장 최근의 유효한 Checkpoint 에 가장 정확한 정보가 있고, 마지막으로 갈수록 최근의 데이터가 저장되어 있다.

이를 바탕으로 우선 Checkpoint Superblock 을 검색하기 위해 signature "NXSB"를 검색하였다.

그 중 가장 마지막 superblock 의 data 를 가져왔다. Data 는 signature 이전의 header 정보도 포함하기 위해 NXSB signature 를 기준으로 앞의 0x20 만큼, 뒤로 0x1000 만큼, 총 0x1020 길이를 복사하였다. 그리고 [사진 6]의 위치에 붙여넣었다.

0C80:4FF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:5090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:50A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:50B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:50C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:50D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0C80:50E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

[사진 7] Checkpoint Superblock data 복사 후 붙여넣기

저장 후 다시 APFS.bt 템플릿을 실행시켰다.

[WHS-2] .iso

Name	Value	Start	Size	Type
> gpt_header	200h	200h		struct GPT
> table[128]	400h	4000h		struct GPTPARTITIONTABLE
✓ apfs	C805000h	0h		struct APFS

➡

Name	Value	Start	Size	Type
> gpt_header	200h	200h		struct GPT
> table[128]	400h	4000h		struct GPTPARTITIONTABLE
✓ apfs	C805000h	0h		struct APFS
✓ csb	C805000h	1000h		struct obj
> hdr	C805000h	20h		struct obj_header
> body	C805020h	548h		struct container_superblock
✓ cp	C806000h	2C000h		struct CheckpointDesc
> checkpoint...	C806000h	2C000h		struct obj
✓ valid_csb	C831000h	1000h		struct obj
> hdr	C831000h	20h		struct obj_header
> body	C831020h	548h		struct container_superblock

[사진 8] 수정 후 APFS 템플릿 실행 결과창

이전에는 없었던 APFS 블록에 Block이 생긴 것을 확인할 수 있다.

하지만, 템플릿이 완전하게 실행되지 않았고, 오류가 발생했다.

```
paddr invalid in obj_ref!! Not defining omap!!
*ERROR Line 2023: Variable 'omap' is not a member of the structure.
```

[사진 9] AFPS.bt template 실행 중 오류

paddr값이 invalid하여 Omap 식별에 오류가 생긴 것을 확인했다.

✓ entries[1]		D8E3038h	4h	struct omap
✓ entries	OID 1026, XID 110	D8E3038h	4h	struct omap
key_offset	16	D8E3038h	2h	short
data_offset	32	D8E303Ah	2h	short
✓ key_header		D8E3208h	10h	struct omap
oid	1026	D8E3208h	8h	uint64
xid	110	D8E3210h	8h	uint64
✓ val		D8E3FE0h	10h	struct omap
flags	(16) CRYPTO_GENERATION	D8E3FE0h	4h	uint
size	16	D8E3FE4h	4h	uint
paddr	68719476752	D8E3FE8h	8h	uint64

[사진 10] paddr

csb-omap-body-tree-body-entries 경로로 paddr를 확인하였다.

Paddr 은 Physical Address로 여기서 omap의 물리적 주소를 알려주어야 하는데, 4,096byte인 이 블록을 생각해봤을 때 [사진10]의 paddr은 불가능한 숫자이다.

Omap의 paddr을 찾기 위해 Volume Superblock을 찾아보았다. Volume Superblock의 시그니처인 APSB를 검색해보았다.

Find Results	
Address	Value
Found 89 occurrences of 'APSB'.	
D72E020h	APSB
D73E020h	APSB
D745020h	APSB

[사진 11] APSB 검색 결과창

89개의 결과를 찾았다.

tree		D8E3000h	1000h
> hdr		D8E3000h	20h
> body		D8E3020h	1Ch
node_type	(6) BTNODE_LEAF, BTNODE_FIXED_KV_SIZE	D8E3020h	2h
level	0	D8E3022h	2h
entry_count	1	D8E3024h	4h
table_space_off	0	D8E3028h	2h
table_space_len	448	D8E302Ah	2h
free_space_off	32	D8E302Ch	2h
free_space_len	3488	D8E302Eh	2h
key_free_list_off	0	D8E3030h	2h
key_free_list_len	16	D8E3032h	2h
val_free_list_off	16	D8E3034h	2h
val_free_list_len	16	D8E3036h	2h
entries[1]		D8E3038h	4h
entries	OID 1026, XID 110	D8E3038h	4h

[사진 12] omap - entry_count

하지만, 이 tree의 body는 Volume Superblock인데 entry count가 1개이기 때문에 실질적인 볼륨은 1개이다. 따라서 [사진11]의 89개 중 1개의 올바른 볼륨을 찾아야 한다.

[사진 7]을 찾을 때처럼 우선 가장 최근의 Superblock들을 찾아야 한다. 또한, checksum값이 정상적이어야 한다. 이렇게 찾아보았을 때 Last_mod_time이 가장 큰 값을 가지는 총 10개의 후보가 나왔다.

출력값을 바로 paddr에 넣어볼 수 있게 uint64로 나타낸 후보: 4190, 4222, 4233, 4240, 4246, 4262, 4275, 4275, 4285, 4313, 4316

그 중 가장 마지막에 저장되어 있는 것을 가장 최근 값으로 추정하고 4316부터 입력해보았다.

템플릿 실행 결과, 정상적으로 실행되었다.

Name	Value	Start	Size	Type	
> gpt_header	200h	200h		struct GPT	
> table[128]	400h	4000h		struct GPTPARTITIONTABLE	
> apfs	C805000h	10DD000h		struct APFS	
> csb	C805000h	1000h		struct obj	
> cp	C806000h	2C000h		struct CheckpointDesc	
> valid_csb	C831000h	1000h		struct obj	
> vol	D8E1000h	1000h		struct obj	

entries[1]	OID 1026, XID 110	4h	struct omap_node_entry
entries		4h	struct omap_node_entry
key_offset	16	2h	short
data_offset	32	2h	short
key_header		10h	struct omap_key
oid	1026	8h	uint64
xid	110	8h	uint64
val		10h	struct omap_val
flags	(16) CRYPTO_GENERATION	4h	uint
size	16	4h	uint
paddr	4316	8h	uint64
omap		1000h	struct obj
> hdr		20h	struct obj_header
> body		400h	struct apfs_superblock

Template executed successfully.

[사진 13] paddr 오류 해결 후

[WHS-2] .iso

Paddr값을 입력하니

1. 기존에 오류 나던 Template실행이 성공적으로 실행됐다.
2. Volume Superblock이 생성되었다.
3. Csb-body-omap-tree-omap가 생성되었다.

```
> vol_uuid          00000000-0000-0000-0000-00...
  last_mod_time      07/15/2021 08:06:09
  vol_flags           (8) onekey
> formatted_by
> modified_by[8]
> volname[256]       DataBackup
  next_doc_id         3
  apfs_role            (0) NONE
```

[사진 14] Volume Superblock 정보

새로 인식된 Volume Superblock에서 omap 관련 정보들을 자세히 살펴보았다.

```
tree
> hdr
  body
    node_type          (7) BTNODE_ROOT,BTNODE_L
    level              0
    entry_count         0
    table_space_off     0
    btree_information
      bt_flags          (18) SEQ_INSERT,PHYSICAL
      bt_node_size      4096
      bt_key_size        16
      bt_val_size        16
      bt_longest_key     16
      bt_longest_val     16
      bt_key_count       11
      bt_node_count      1
```

[사진 15] Volume Superblock 내 tree

Tree 내 body에서 식별된 entry_count는 0인 반면, btree_information에서 key_count가 11이다. 정상적으로 식별됐다면 entry_count와 Key_count는 개수가 같아야 한다. 따라서 entry_count를 11로 바꿔주었다.

```
body
  node_type          (7) BTNODE_ROOT,BTNODE_LEA...
  level              0
  entry_count         11
  table_space_off     0
  table_space_len     448
  free_space_off      192
  free_space_len      3168
  key_free_list_off    48
  key_free_list_len    16
  val_free_list_off    64
  val_free_list_len    16
  entries[11]
    > entries[0]      OID 1028, XID 109
    > entries[1]      OID 1030, XID 108
    > entries[2]      OID 1031, XID 107
    > entries[3]      OID 1033, XID 106
    > entries[4]      OID 1034, XID 109
    > entries[5]      OID 1035, XID 109
    > entries[6]      OID 1037, XID 106
    > entries[7]      OID 1038, XID 102
    > entries[8]      OID 1041, XID 102
    > entries[9]      OID 1042, XID 108
    > entries[10]     OID 1043, XID 109
```

[사진 16] 정상적으로 인식된 entries[11]

2. UUID값 복원

1) CSB SuperBlock - uuid

apfs		C805000h	0h	struct APFS	
csb		C805000h	1000h	struct obj	
hdr		C805000h	20h	struct obj_header	
cksum	7559093634750793190	C805000h	8h	uint64	
oid	1	C805008h	8h	uint64	
xid	110	C805010h	8h	uint64	
type	obj_type_container_superblock (1)	C805018h	2h	enum obj_type	
flags	OBJ_EPHEMERAL	C80501Ah	2h	ushort	
subtype	0	C80501Ch	2h	enum obj_type	
body		C805020h	548h	struct container_superblock	
> magic[4]	NXSB	C805020h	4h	char	
block_size	4096	C805024h	4h	uint	
block_co...	937206	C805028h	8h	uint64	
features_0	0	C805030h	8h	uint64	
read_onl...	0	C805038h	8h	uint64	
incompa...	2	C805040h	8h	uint64	
> uuid	00000000-0000-0000-0000-000000000000	C805048h	10h	struct Uuid	

[사진 17] CSB(Checkpoint SuperBlock) Header와 Body

uuid값이 비어있다. Uuid(Universally Unique Identifier)는 고유한 식별자값인데 0으로 구성되어 있는 것은 말이 되지 않는다.

다른 checkpoint들도 모두 열어 uuid값이 존재하는지 찾아보았다.

apfs		C805000h	0h	struct APFS	
> csb		C805000h	1000h	struct obj	NX xid=110
> cp		C806000h	2C000h	struct Checkpo...	
> checkpoint_desc_nx[44]		C806000h	2C000h	struct obj	
> checkpoint_desc_nx[0]		C806000h	1000h	struct obj	
> checkpoint_desc_nx[1]		C807000h	1000h	struct obj	NX xid=89
> checkpoint_desc_nx[2]		C808000h	1000h	struct obj	
> checkpoint_desc_nx[3]		C809000h	1000h	struct obj	NX xid=90
> hdr		C809000h	20h	struct obj_head...	
> body		C809020h	548h	struct containe...	
> magic[4]	NXSB	C809020h	4h	char	
block_size	4096	C809024h	4h	uint	
block_count	937206	C809028h	8h	uint64	
features_0	0	C809030h	8h	uint64	
read_only_compati...	0	C809038h	8h	uint64	
incompatible_feat...	2	C809040h	8h	uint64	
> uuid	00000000-0000-0000-0000-000000000000	C809048h	10h	struct Uuid	

[사진 18] uuid값이 없는 checkpoint_desc_nx

확인하면서 Checkpoint_desc_nx[홀수]는 uuid값을 나타내었고, [짝수]는 uuid값을 아예 나타내지 않았다.

[WHS-2] .iso

▼ checkpoint_desc_nx[21]				
> hdr	C81B000h	20h	struct obj_head...	
▼ body	C81B020h	548h	struct containe...	
> magic[4]	C81B020h	4h	char	
block_size	C81B024h	4h	uint	
block_count	C81B028h	8h	uint64	
features_0	C81B030h	8h	uint64	
read_only_compati...	C81B038h	8h	uint64	
incompatible_feat... 2	C81B040h	8h	uint64	
> uuid	C81B048h	10h	struct Uuid	
next_oid	C81B058h	8h	uint64	

[사진 19] uuid값이 존재하는 checkpoint_desc_nx[21]

▼ checkpoint_desc_nx[43]				
> hdr	C831000h	20h	struct obj_head...	
▼ body	C831020h	548h	struct containe...	
> magic[4]	C831020h	4h	char	
block_size	C831024h	4h	uint	
block_count	C831028h	8h	uint64	
features_0	C831030h	8h	uint64	
read_only_compati...	C831038h	8h	uint64	
incompatible_feat... 2	C831040h	8h	uint64	
> uuid	C831048h	10h	struct Uuid	
next_oid	C831058h	8h	uint64	

[사진 20] uuid값이 포함된 checkpoint_desc_nx[43]

Checkpoint_desc_nx[홀수] 번호들을 열어보던 중 uuid값이 포함된 Checkpoint 2개를 찾았다.

uuid값은 고유성을 가진 식별자이기 때문에 위 두 개의 값과 Checkpoint Superblock의 uuid값도 같을 것이다.

2) CSB SuperBlock - uuid

Volume Superblock에도 uuid값이 00000임을 확인했다. 수정이 필요하다.

위에서 omap의 paddr값을 찾기 위해 사용했던 것처럼, Volume Superblock의 시그니처인 APSB를 검색하여 89개의 블록을 다시 한 번 찾았다.

0D74:50F0	ED 26 AA A3	F3 40 4A A1	B9 C4 62 94	DC 84 AD BF
0D76:D0F0	ED 26 AA A3	F3 40 4A A1	B9 C4 62 94	DC 84 AD BF
0D9A:E0F0	ED 26 AA A3	F3 40 4A A1	B9 C4 62 94	DC 84 AD BF
0DAF:00F0	ED 26 AA A3	F3 40 4A A1	B9 C4 62 94	DC 84 AD BF
0DCA:30F0	ED 26 AA A3	F3 40 4A A1	B9 C4 62 94	DC 84 AD BF
0DD1:00F0	ED 26 AA A3	F3 40 4A A1	B9 C4 62 94	DC 84 AD BF

[사진 21] 동일한 블록의 uuid값 중 일부 사진

89개 모두 uuid값 위치에 같은 값을 가지고 있었다. Contain Superblock의 uuid값처럼 Volume Superblock 또한 같은 uuid를 사용할 것이다.

```

0D8E:1010 6E 00 00 00 00 00 00 00 0D 00 00 00 00 00 00 00 n.....
0D8E:1020 41 50 53 42 00 00 00 00 02 00 00 00 00 00 00 00 APSB.....
0D8E:1030 00 00 00 00 00 00 00 00 05 00 00 00 00 00 00 00 .....
0D8E:1040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0D8E:1050 00 00 00 00 00 00 00 00 41 01 00 00 00 00 00 00 .....A.....
0D8E:1060 05 00 00 00 00 00 00 00 06 00 00 00 E2 00 45 12 .....ä.E.
0D8E:1070 01 00 00 00 02 00 00 00 02 00 00 40 02 00 00 40 .....@...@
0D8E:1080 D4 10 00 00 00 00 00 00 04 04 00 00 00 00 00 00 Ö.....
0D8E:1090 BA 10 00 00 00 00 00 00 20 0F 00 00 00 00 00 00 °.....
0D8E:10A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0D8E:10B0 E9 00 00 00 00 00 00 00 4D 00 00 00 00 00 00 00 é.....M.....
0D8E:10C0 1B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0D8E:10D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0D8E:10E0 1B 03 00 00 00 00 00 00 E9 01 00 00 00 00 00 00 .....é.....
0D8E:10F0 ED 26 AA A3 F3 40 4A A1 B9 C4 62 94 DC 84 AD BF i&*&ó@J; 'Äb"Ü„-¿
0D8E:1100 20 F1 6E 98 4F E8 91 16 08 00 00 00 00 00 00 00 ñn~Oè'.....

```

Name	Value	Start
> vol_uuid	ed26aaa3-f340-4aa1-b9c4-6294dc84adbf	D8E10F0h

[사진 22] vol_uuid 수정

어느 정도 수정과 복구를 마쳐 checksum을 검증해야 한다. 또한, 암호화되어있던 부분을 복호화 해줘야 하는데 010editor에서도, cmd에서도, visual studio에서도 작성해둔 코드를 USB-002.dd에 실행되지 않아서 더 이상 풀이는 어려웠다.

5. Flag

6. 별도 첨부

<세부 문제>

1. Repair the USB and then find password hint for the USB. (200 points)
2. What is the access password for the file server? (50 points)
3. Develop a tool to repair damaged parts of the given USB image file by analyzing the file system structure. (If library is required, include library or installation file) (150 points)

Tool requirements

- Input: USB.dd
- Output: Repaired USB image file

7. Reference

- [010 Editor - Binary Template Information \(sweetscape.com\)](https://www.sweetscape.com/010editor/)
- [APFS: Containers and volumes – The Eclectic Light Company](#)
- [APFS | SpringerLink](#)
- [CC BY 4.0 Deed | Attribution 4.0 International | Creative Commons](#)
- [macOS APFS 구조 \(tistory.com\)](#)