[BOB6]DF12_Tech_05_문의성

디지털 포렌식 트랙 6 기 문의성

작성일자: 2018. 02. 25.

목차

Ⅰ 과제 내용	2
1. 개발환경	2
2. 사용법	2
3. 소스코드	3
1) 레지스트리	3
a) USBSTOR	3
b) 마운트명	4
c) 볼륨명	4
2) setupapi	5
3) 이벤트 로그	6
4) shellbag	6
5) Csv 출력	8
4. 동작 원리	9
Ⅱ 정리 및 요약	10

I 과제 내용

1. 개발환경

해당 프로그램은 Windows 10 OS 에서 JetBrains PyCharm Community Edition 2017.2.4 으로 작업을 진행했습니다. Python 환경은 Python3.6 Version 을 이용했습니다.

Python 3.6 Version

2. 사용법.

1) Python3 구동을 위해 필요한 다음 모듈을 설치해야 합니다.

우선 pip 버전이 9 버전이어야 아래의 piexif 를 패키지를 받을 수 있습니다.

Pip install winreg
Pip install python-evtx
Pip install lxml
Pip install csv

2) Python3 를 통해서 main.py 를 실행합니다.

실행하면 자동으로 파싱이 진행되고 output.csv 파일이 결과로 main.py 와 같은 경로에 추출 됩니다.

3. 소스코드

1) 레지스트리 파싱

클래스를 통하여 시리얼 값을 통해 객체를 먼저 만들고 시리얼 값을 바탕으로 추가로 파싱된 값들을 기존의 객체를 찾아 값을 추가하는 형식으로 진행했습니다.

```
11  class USB_Class:
12      def __init__(self):
13          self.serial = None
14          self.s_time = None
15          self.f_stime = []
16          self.D_name = None
17          self.V_name = None
18          self.M_name = None
19          self.f_ftime = []
20          self.evt_life = []
21          self.execute_t = []
22          self.execute_n = []
23          self.execute_num = []
```

a) USBSTOR

```
for i in range(1024):
    try:
        keyname = EnumKey(varkey, i)
        varSubkey2 = "%s\\%s" % (varSubkey, keyname)
        varkey2 = OpenKey(varReg, varSubkey2)

        try:
            for j in range(1024):
                keyname3 = EnumKey(varkey2, j)
                varSubkey3 = "%s\\%s" % (varSubkey2, keyname3)
                varkey3 = OpenKey(varReg, varSubkey3)
                try:
                    for k in range(1024):
                        index = 0
                        if (index_serial(keyname3) == -1):
                            index = len(usb_barket)
                            usb_barket.append(keyname3)
                            a = USB_Class()
                            Usb_class_barket.append(a)
                        else:
                            index = index_serial(keyname3)

                        n, v, t = EnumValue(varkey3, k)
                        if (n == "FriendlyName"):
                            print("index", index)
                            Usb_class_barket[index].set_usbstor(keyname3, v)
                except:
                    print("3")
                CloseKey(varkey3)
        except:
            print("2")
```

USBSTOR 을 winreg 를 통해서 파싱하는 부분 입니다. USBSTOR 에서는 key 값을 바탕으로 시리얼 값을 획득하고 key 값 중 FriendlyName 을 통해서 장지 드라이버 명을 획득 할 수 있습니다. 처음에 USBSTOR 을 파싱한 이유는 이후에 이벤트 로그 파싱 때도 시리얼 값을 이용해서 합쳤기 때문입니다.

b) 마운트명

```
varSubkey = 'SOFT▼ARE▼▼Microsoft▼▼▼Windows Portable Devices▼▼Devices'
varReg = ConnectRegistry(None, HKEY_LOCAL_MACHINE)
varkey = OpenKey(varReg, varSubkey)

for i in range(1024):
    try:
        keyname = EnumKey(varkey, i)
        varSubkey2 = "%s▼▼%s" % (varSubkey, keyname)
        varkey2 = OpenKey(varReg, varSubkey2)
        try:
            for j in range(1024):
                keys = keyname.split('#')[-2]
                keys = keys.lower()

                index = index_serial(keys)
                if (index == -1):
                    index = index_serial(keys.upper())
                n, v, t = EnumValue(varkey2, j)
                if (index != -1 and n == 'FriendlyName'):
                    Usb_class_barket[index].set_M(v)

        except:
            print("2")
    except:
        break
    CloseKey(varkey2)
```

마운트 명은 해당 레지스트리 경로로 이동하여 FreindlyName 을 바탕으로 획득 했습니다. 이 때 key 값을 split 을 통해 문자열을 잘라내어 시리얼 값 부분만 추출하고 index_serial(해당 키 값을 가지는 객체를 찾아주는 함수)를 통해서 기존에 생성했던 객체에 마운트 명 정보를 추가 했습니다.

c) 볼륨명

볼륨명 역시 마운트명과 마찬가지로 해당 레지스트리 경로를 바탕으로 키 값 문자열을 잘라내어 시리얼 값을 추출하고 해당 시리얼 값을 가지는 객체를 찾아 객체에 볼륨명 값을 추가 했습니다.

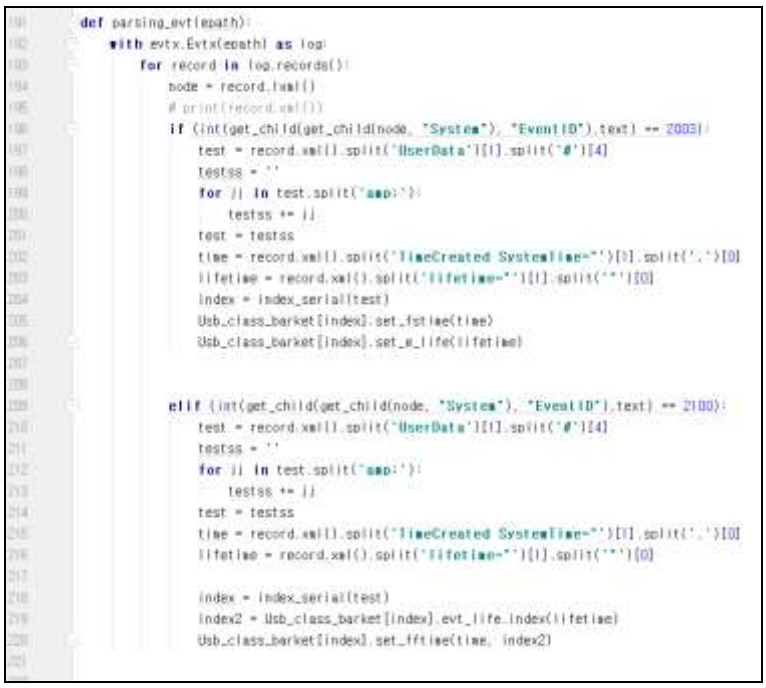
```
varSubkey = 'SYSTEM▮▮▮ControlSet001▮▮Enum▮▮▮pdBusEnumRoot▮▮UMB'
varReg = ConnectRegistry(None, HKEY_LOCAL_MACHINE)
varkey = OpenKey(varReg, varSubkey)

for i in range(1024):
    try:
        keyname = EnumKey(varkey, i)
        varSubkey2 = "%s▮▮%s" % (varSubkey, keyname)
        varkey2 = OpenKey(varReg, varSubkey2)
        try:
            for j in range(1024):
                keys = keyname.split('#')[-2]
                keys = keys.lower()
                index = index_serial[keys]
                if (index == -1):
                    index = index_serial[keys.upper()]
                n, v, t = EnumValue(varkey2, j)
                if (index != -1 and n == 'FriendlyName'):
                    Usb_class_barket[index].set_v(v)
        except:
            print("2")
    except:
        break
    CloseKey(varkey2)
```

2) setupapi

```
f = open('C:▮▮▮indows▮▮inf▮▮setupapi.dev.log', 'r')
lines = f.readlines()
kkk = 0
# print('ssss')
no_overlap = []
indexss = None
for line in lines:
    if (kkk == 2):
        # print('aaabbbbb')
        time = line.split("start ")[1].split('.')[0]
        Usb_class_barket[indexss].set_stime(time)
        kkk = 0
    if (line.find(">>>  [Device Install (Hardware initiated) - USBSTOR") != -1):
        serial = line.split("▮▮")[-1].split("]")[0]
        if (serial in no_overlap):
            continue
        no_overlap.append(serial)
        indexss = index_serial[serial]
        # print(indexss)
        kkk = 2
```

Setupapi 은 >>> [Device install (Hardware initialted) – USBSTOR 문구를 통해서 최초 연결 시 기록되는 부분을 찾고 USB 최초 연결 시간을 파싱한 후 setupapi 에서 시리얼 값을 문자열 자르기를 통해서 얻은 후 시리얼 값을 바탕으로 매칭되는 객체를 찾아 최초 연결 시각을 기록 했습니다.

3) 이벤트 로그



```
def parsing_evt(epath):
    with evtx.Evtx(epath) as log:
        for record in log.records():
            node = record.lxml()
            # print(record.xml())
            if (int(get_child(get_child(node, "System"), "EventID").text) == 2003):
                test = record.xml().split('UserData')[1].split('#')[4]
                testss = ''
                for ji in test.split('amp;'):
                    testss += ji
                test = testss
                time = record.xml().split('TimeCreated SystemTime=''')[1].split(',')[0]
                liftime = record.xml().split('liftime=''')[1].split('''')[0]
                index = index_serial(test)
                Usb_class_barket[index].set_fstime(time)
                Usb_class_barket[index].set_e_life(liftime)


            elif (int(get_child(get_child(node, "System"), "EventID").text) == 2100):
                test = record.xml().split('UserData')[1].split('#')[4]
                testss = ''
                for ji in test.split('amp;'):
                    testss += ji
                test = testss
                time = record.xml().split('TimeCreated SystemTime=''')[1].split(',')[0]
                liftime = record.xml().split('liftime=''')[1].split('''')[0]

                index = index_serial(test)
                index2 = Usb_class_barket[index].evt_life_index(liftime)
                Usb_class_barket[index].set_fftime(time, index2)
```

다음 부분은 이벤트 로그 파싱 부분으로 dirver 이벤트 로그 경로로 접근하여 드라이버 장치가 연결 시 입력되는 이벤트로그인 2003 과 연결 해제 시 입력되는 이벤트 로그 2100 을 바탕으로 각각의 시간 값을 가져 왔습니다. 이 때 이벤트 로그에 기록되는 시리얼 값과 liftetime 을 바탕으로 연결 해제 시간을 쌍을 지어 객체에 기록 했습니다.

4) shellbag

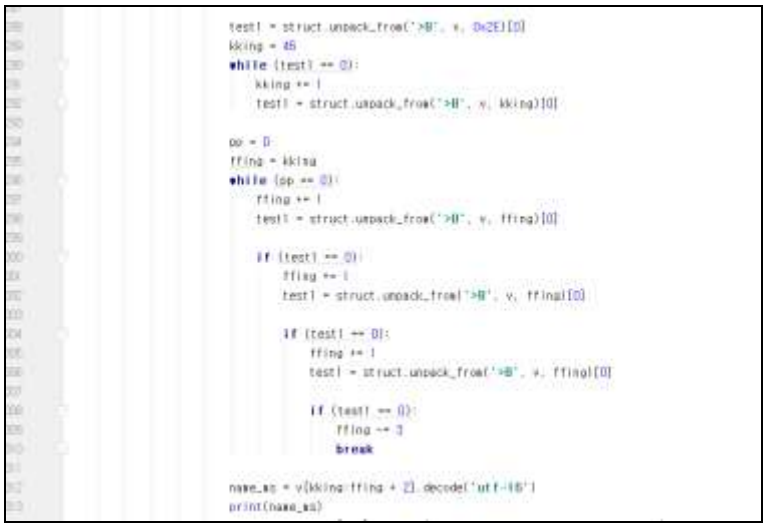
이 부분은 레지스트리 현재 유저 Hive 부분의 초입 파싱 부분입니다. shellbag 에서 C 드라이브가 아닌 다른 드라이브에서 실행된 파일만 파싱 하기 위해서 우선 Shell₩BagMRU 경로에 가면 1 번에 실행 되었던 각각의 볼륨명이 키 값으로 존재 하기 때문에 해당 부분을 파싱 했습니다. 이 key 값을 바탕으로 현재 C 드라이브를 제외한 각각의 다른 드라이브를 추출하며 얻어낸 키 값을 바탕으로 circuit 을 통해서 레지스트리 구조를 재귀 함수를 통해서 구조 상 안쪽으로 들어가며 파싱을 진행했습니다.

```
for i in range(1024):
    try:
        keyname = EnumKey(varkey, i)
        varSubkey2 = "%s\\%s" % (varSubkey, keyname)
        varkey2 = OpenKey(varReg, varSubkey2)
        if (keyname == 'I'):

            try:
                for j in range(1024):

                    n, v, t = EnumValue(varkey2, j)
                    if (n != 'MRUListEx' and n != 'NodeSlot'):
                        v = v.decode('utf-8')
                        v = v.split('\\')[0].split('/')[1]
                        if (v == 'C:' or v == 'c:'):
                            continue

                        circuit(varkey2, varSubkey2, varReg, v, n)
            except:
                print("")
    except:
        break
    CloseKey(varkey2)
```

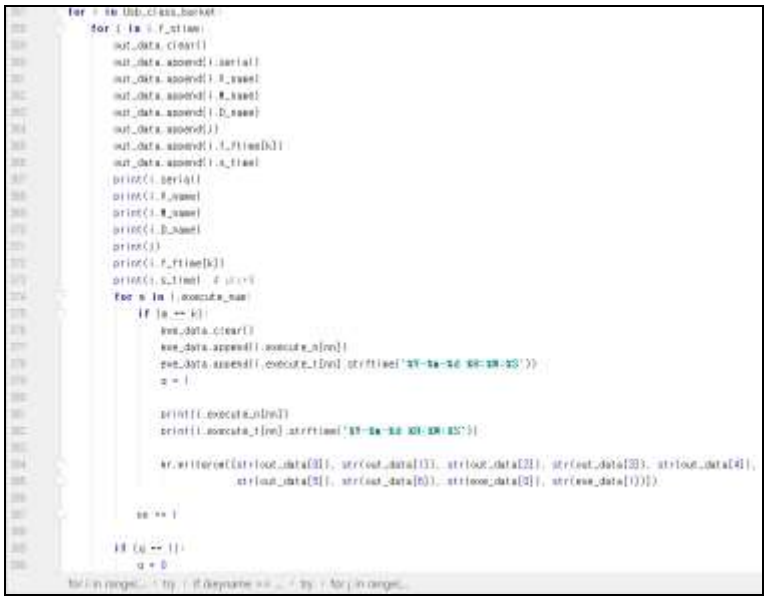
다음으로 아래 시진과 같이 쉘 백을 안의 키 값들의 수정 시간을 가져와 파일이 실행된 시간을 파싱 했습니다. 레지스트리 키 값의 수정 시간은 QueryinfoKey 를 통해서 추출했고 check_time 함수를 통해서 기존에 기록된 USB 연결 해제 시간 쌍과 비교하여 적절한 객체를 판별했고 실행 파일들을 해당 USB 와 매칭했습니다.

```
            if (strkeyname) == str(ann)):
                for jj in range(1024):
                    n, v, t = EnumValue(varkey, jj)

                    if (n != 'MRUListEx' and n != 'NodeSlot'):
                        cop = 1

                        ts = QueryInfoKey(varkey)[2]
                        # print('77777777')
                        dt = dt_from_win32_ts(ts)
                        # print('88888888')
                        indexxx = -1
                        iii = 0
                        for i in Usb_class_barket:
                            # print(type(dt))
                            indexxx = check_time(i, dt)  # dt.strftime('%Y-%m-%d %H:%M:%S')
                            # print('indexxx', indexxx)
                            if (indexxx != -1):
                                break
                            iii += 1
                        print(indexxx)
```

마지막으로 Shellbag 의 filename 부분은 shellbag 구조상 0x2E 부분부터 첫 0x00 이 아닌 값으로부터 마지막 0x000000 전까지 부분이 FIleName 을 가리켰고 이 부분은 가변적이기 때문에 반복문을 통해서 파일 name 의 첫번째 부분과 마지막 부분의 index 를가져오고 전체 shellbag 데이터에서 Filename 의 첫번째 부분부터 마지막 부분까지 decode('utf-16')을 통해 FileName 을 획득했습니다.



5) Csv 출력



마지막으로 값을 저장하고 있는 객체들을 불러와서 csv 파일 형태로 값을 출력하고 있습니다.

4. 동작 원리

소스 코드의 동작 원리를 정리하면 다음과 같습니다.

우선 레지스트리 USBSTOR 부분을 파싱하여 Serial 값과 장치 드라이브 명을 획득하여 USB_Class 객체에 넣어줍니다. 그리고 마운트명과 볼륨명을 각각의 레지스트리 경로에가서 획득하고 여기서 키 값을 일부 잘라내어 시리얼 값을 획득한 후 기존에 생성한 객체의 시리얼 값과 비교하여 적절한 객체에 마운트 명과 볼륨명을 추가합니다.

다음으로 setupapi 의 최초 장치 드라이버 연결 시 출력되는 문구를 바탕으로 find 를 진행하여 최초 연결 시각을 찾고 마찬가지로 시리얼 값을 문자열 자르기를 통해서 획득 한 후 기존의 객체 시리얼 값과 비교하여 적절한 객체에 최초 연결 시각을 추가합니다.

다음으로 장치 드라이브 관련한 이벤트 로그를 파싱하여 Event ID=2003 으로부터 시리얼 값과 연결 시각 그리고 lifetime 을 가져옵니다. 이 때 lifetime 을 가져오는 경우는 같은 USB 로 여러 번 연결 해제 했을 경우 시리얼 값만으로는 구분이 어렵기 때문입니다. 즉 시리얼 값과 lifetime 을 통해서 연결 시각을 객체에 추가해주고 마찬가지로 Event ID=2100 을 통해서 연결 해제 시각을 추가합니다.(lifetime 은 2003,2100 쌍으로 존재합니다!)

마지막으로 USB 내 실행 여부를 판단하기 위해 Shellbag 을 파싱했습니다. ShellBag 의 경우 Shell₩BagMUR 경로에 순차적으로 0 부터 값들이 기록되며 1 번 에 각각의 볼륨 루트가 키 값으로 적혀있습니다. 이 부분을 통해 C 드라이브를 제외한 드라이브 값을 획득하고 해당 키 값을 바탕으로 재귀 함수를 통해서 구조상 안쪽을 반복해서 파싱해 가는 방법으로 Shellbag 을 파싱했습니다. ShellBag 에서는 실행한 파일 Full_Path 와 실행된 시각을 얻을 수 있었고 실행된 시각을 바탕으로 기존에 기록한 USB 연결 해제 시간과 비교하여 적절한 USB 에 값을 위치 시켰습니다.

II 정리 및 요약

USB 를 통한 정보 유출관련해서 과제를 진행하였고 필요한 정보들을 파싱해와서 종합하는 부분을 진행했습니다. 특히 Shellbag 을 진행하는 과정에서 레지스트리의 전반적인 구조와 Shellbag 의 저장 구조 등의 이해가 필요했는데 과제를 통해 직접 파싱함으로써 해당 부분에 대한 이해가 잘 될 수 있었던 것 같습니다!!!

감사합니다!!