디지털 포렌식을 위한 동작 중인 메모리에서의 파일 정보 수집

박진규, 이제훈, 김상욱
경북대학교 전자전기컴퓨터학부
e-mail : jkpark@woorisol.knu.ac.kr

A Live Acquisition of File Information from Memory for Digital Forensic

Jinkyu Park, Jaehun Lee, Sangwook Kim
School of Electrical Engineering and Computer Science
Kyungpook National University

요 약

기존의 디지털 포렌식 기술은 하드 디스크 등에서 증거 자료를 수집하는 기술을 연구해 왔다. 하지만 최근 루트킷 등 악성 프로그램의 은닉 기술 발달로 디스크 악성 프로그램의 흔적이 남지 않게 되었고, 디스크 용량의 기하급수적인 증가로 필요한 증거 자료를 찾기 위해 디스크를 탐색하는 시간이 증가하였다. 메모리 포렌식 기술은 기존의 디지털 포렌식의 단점을 보완하는 새로운 연구분야로, 동작 중인 시스템에서 메모리 내부의 정보를 수집하고 분석하는 데 초점을 맞추고 있다. 본 논문에서는 메모리 포렌식 기법으로 수집할 수 있는 자료인 파일 정보를 동작 중인 메모리에서 수집하고 분석하는 방법에 대해 알아본다.

1. 서론

기존의 디지털 포렌식 기술은 하드 디스크 등에서 증거 자료를 수집하는 기술을 연구해 왔다. 파일 형식에 따는 시그니쳐를 탐색하거나 시스템의 로그 등을 분석해 사고가 발생한 시스템의 기록을 수집하는 형태의 디지털 포렌식 기술로 사고 순서나 사용자의 행동에 대한 증거를 제시했다.

그러나 최근 하드 디스크 용량의 기하급수적인 증가로 디스크 내부에서 디지털 포렌식에 필요한 증거 자료를 수집하는 데 필요한 시간이 증가하였다. 또한 최근에 발표된 루트킷 등의 악성 프로그램은 탐지되는 것을 막기 위해 메모리 상에만 존재하는 방식을 사용한다. 악성 프로그램이 커널 내부를 변조시켜 임의의 프로세스 목록이나 열린 파일 목록 등을 숨길 경우 일반적인 방법으로는 시스템 내부에서 악성 프로그램의 존재 여부를 판단하는 데 많은 어려움이 있다.

따라서 실행 중인 컴퓨터에서 사용자가 실행한 동작을 파악하기 위해 메모리에 존재하는 정보를 수집하고 분석하는 메모리 포렌식에 관한 연구가 2005 년 Mariusz Burdach[1] 를 기준으로 시작되었다. 컴퓨터의 전원이 꺼지면 사라지는 휘발성 정보인 메모리 내부의 정보들은 컴퓨터가 동작 중일 때 항상 변경되기 때문에[2] 포렌식 과정에서 중요하게 사용할 수 있는 증거로는 볼 수 없고, 사용자의 행동을 예측하는 단서로서의 의미가 강하다. 현재까지 연구된 메모리 포렌식 기법으로는 메모리 내부의 정보를 종류에 따라 실시간으로 수집하거나, 메모리 자체를 소프트웨어적인 방법 또는 하드웨어적인 방법으로 덤프한 후 디스크 포렌식에서 사용하는 시그니쳐 탐색 방법을 응용해 필요한 정보를 수집하는 방법 등이 있다[3].

본 논문에서는 윈도우 운영체제에서 메모리 포렌식 기법으로 수집할 수 있는 자료인 파일 목록에 대한 정보를 동작 중인 메모리에서 수집하고 분석하는 방법에 대해 알아본다. 시스템 내부 정보를 수집하기 위해 조사자가 실행하는 프로세스의 권한을 상승시키고, 해당 프로세스의 핸들을 수집한 후 파일 정보를 추출하는 기법을 사용해서 시스템의 프로세스가 사용중인 열린 파일에 대한 정보를 수집한다.

본 논문의 구성은 다음과 같다. 2 절에서는 메모리 포렌식과 관련된 다양한 연구 결과에 대해 먼저 살펴본다. 3 절에서는 제안하는 동작 중인 메모리에서 파일 정보 수집 방법에 대해 기술한다. 4 절에서는 제안한 기법의 개발에 대해 알아보고 5 절에서 결론과 향후 연구 방향을 제시한다.

2. 관련 연구

본 절에서는 메모리에서 포렌식에 필요한 정보를 수집하는 방법에 대해 현재까지 연구된 내용을 기술한다. 2005 년에 Mariusz Burdach 는 리눅스의 메모리

정보를 수집하는 도구를 발표하였다. 메모리에 올라가 있는 파일과 프로그램의 내용을 복구하고, 프로세스 목록을 수집하는 것이 주 기능이다[1]. Andreas Schuster 는 윈도우 메모리 할당에 사용되는 pool 구조체의 동작 방식이 메모리 포렌식에 미치는 영향에 대해 연구하고, 메모리에서 상대적으로 변경되는 시간이 느린 pool 내부의 정보를 활용하는 방안을 제안했다[4]. 또한 Brendan Dolan-Gavitt 은 윈도우 시스템에서 페이지 디렉토리를 탐색하여 가상 메모리와 물리 메모리 사이의 매핑을 담당하는 Virtual Address Descriptor 정보를 수집하고 이를 통해 메모리 영역에 대한 포괄적인 정보를 획득하는 연구를 진행했다[5].

메모리 포렌식에 관한 연구 중에는 메모리나 스왑 영역의 정보 전체를 디스크로 덤프하거나, 메모리 덤프에서 포렌식에 필요한 단서를 추출하는 기술도 존재한다. Bradly Schatz 는 운영체제 가상화 기술을 응용해 윈도우 시스템 내부에 가상의 운영체제를 실행한 뒤 메모리 정보를 수집하는 방법을 제안했다[6].

Matthieu Suiche 은 디바이스 드라이버를 윈도우 커널에 삽입한 후 커널 단계에서 메모리 정보를 수집하는 도구를 발표했다[7]. 윈도우 운영체제에서 메모리를 나타내는 \\.\PhysicalMemory 파일은 일반적인 사용자 권한으로는 접근이 불가능하지만, 커널 영역에서의 메모리 정보 수집은 권한 상의 제약을 받지 않는다.

Helix 라고 불리는 라이브 시스템 포렌식 툴은 윈도우 메모리를 덤프하기 위해 IEEE 1394 포트를 사용한다[8]. IEEE 1394 포트를 통한 DMA 를 사용하면 사용자 권한과 관계없이 메모리 정보를 수집할 수 있지만, 별도의 컴퓨터와 IEEE 1394 포트로 연결해야 한다는 단점이 존재한다.

3. 동작 중인 메모리에서 파일 정보 수집 방법

본 절에서는 동작 중인 윈도우 시스템에서 사용 중인 메모리에 존재하는 파일에 대한 정보를 수집하는 도구에 대해 기술한다. 파일 정보 수집 도구는 현재 프로세스의 모드를 디버그 모드로 상승시키는 과정과 해당 프로세스의 핸들을 복제하는 과정, 그리고 복제한 핸들로 파일 정보를 추출하는 과정으로 나뉜다.

3.1 프로세스 권한 상승 과정

동작 중인 메모리에서 파일 정보를 수집하기 위해서 먼저 수집 과정을 수행하는 프로세스의 권한을 상승시키는 과정이 필요하다. 상승된 권한을 디버그 모드라고 하고, AdjustTokenPrivileges API 를 사용해서 해당 프로세스의 토큰의 권한을 수정한다. 디버그 모드로 수정하기 위해서는 권한 토큰을 만들 TOKEN_PRIVILEGES 구조체를 설정하고 LookupPrivilegeValue API 를 이용하여 지정된 권한 이름에 대한 지역 단일 식별자(locally unique identifier: LUID)를 획득한다. 그 후 OpenProcessToken API 를 이용하여 프로세스의 토큰 핸들을 얻고 AdjustTokenPrivileges API 를 사용하여 입력 받은 권한으로 프로세스 권한을 바꾼다.

```
BOOL WINAPI LookupPrivilegeValue(
    __in_opt   LPCTSTR lpSystemName,
    __in       LPCTSTR lpName,
    __out      PLUID lpLuid
);
```

LookupPrivilegeValue API 는 해당 시스템에서 특정 이름에 대한 권한을 지정하는 LUID 를 구하는 함수이다. lpName 에 구하는 권한의 이름을 전달하면 lpLuid 에 해당 LUID 가 반환된다.

다른 프로세스의 정보를 획득하기 위해서 lpName 에는 SE_DEBUG_NAME 을 전달한다.

```
BOOL WINAPI OpenProcessToken(
    __in    HANDLE ProcessHandle,
    __in    DWORD DesiredAccess,
    __out   PHANDLE TokenHandle
);
```

구한 LUID 를 적용하기 위해서 현재 프로세스의 토큰을 열기 위해서 OpenProcessToken API 에 현재 프로세스의 핸들을 ProcessHandle 값에 입력하고, DesiredAccess 에 원하는 토큰 형태를 지정한다. 권한을 적용하기 위해서 TOKEN_ADJUST_PRIVILEGES 형으로 지정한다. 그리고 TokenHandle 로 해당 토큰을 반환 받는다.

AdjustTokenPrivileges API 의 형태는 다음과 같다.

```
BOOL WINAPI AdjustTokenPrivileges(
    __in          HANDLE TokenHandle,
    __in          BOOL DisableAllPrivileges,
    __in_opt      PTOKEN_PRIVILEGES NewState,
    __in          DWORD BufferLength,
    __out_opt     PTOKEN_PRIVILEGES PreviousState,
    __out_opt     PDWORD ReturnLength
);
```

TokenHandle 에 OepnProcessToken 에서 획득한 토큰을 전달하고, DisableAllPrivileges 는 NewState 에서 지정하는 권한으로 설정하기 위해서 FALSE 로 지정한다. NewState 에는 지정하는 권한을 담은 TOKEN_PRIVILEGES 형태의 구조체를 전달한다. BufferLength 에는 TOKEN_PRIVILEGES 구조체의 크기를 지정하고, PreviousState 나 ReturnLength 는 0 으로 지정한다.

3.2 프로세스 핸들 복제 과정

프로세스 정보와 핸들 정보를 수집하기 위해서는 NtQuerySystemInformation 과 NtQueryObject API 를 사용한다. NtQuerySystemInformation API 를 이용하여 SystemInformationClass 인자에 SystemHandleInformation 를 전달하여 Handle 정보를 수집한다.

```
typedef struct _SYSTEM_HANDLE
{
```

```
        DWORD           dwProcessId;
        BYTE            bObjectType;
        BYTE            bFlags;
        WORD            wValue;
        PVOID           pAddress;
        DWORD           GrantedAccess;
}SYSTEM_HANDLE;
```

Handle 정보의 전달은 SYSTEM_HANDLE 구조체를 이용하여 전달한다. SYSTEM_HANDLE 구조체에는 dwProcessId 에 프로세스 아이디와 bObjectType 에 핸들의 분류 정보, pAddress 는 FILE_OBJECT 를 가리키는 포인터, wValue 는 이 핸들을 가리키는 값이 반환된다. 이때 전체 핸들 사이즈를 포함하여 전달한다. 따라서 사이즈를 저장할 dwCount 와 SYSTEM_HANDLE 을 포함하는 PPSYSTEM_HANDLE_INFORMATION 구조체를 이용하여 전달한다.

```
typedef struct _SYSTEM_HANDLE_INFORMATION
{
        DWORD           dwCount;
        SYSTEM_HANDLE Handles[1];
}               SYSTEM_HANDLE_INFORMATION,
*PSYSTEM_HANDLE_INFORMATION,
**PPSYSTEM_HANDLE_INFORMATION;
```

먼저 PSYSTEM_HANDLE_INFORMATION 구조체를 임의의 크기의 버퍼를 만들어서 NtQuerySystemInformation 을 호출하여 에러가 뜨면 반환되는 사이즈 크기만큼의 버퍼를 할당하여 다시 호출하여 핸들 정보를 가져온다. PSYSTEM_HANDLE_INFORMATION->dwCount 에는 시스템에서 사용하는 모든 핸들의 개수가 반환되어 이 개수만큼의 핸들 정보를 분석한다. 가져온 핸들의 정보는 OpenProcess 와 DuplicateHandle 을 이용하여 해당 핸들의 오브젝트를 만들어내고 NtQueryObject API 를 이용하여 오브젝트의 정보를 수집한다.

```
HANDLE WINAPI OpenProcess(
  __in  DWORD dwDesiredAccess,
  __in  BOOL bInheritHandle,
  __in  DWORD dwProcessId
);
```

OpenProcess 는 프로세스 정보를 여는 기능을 한다. dwDesiredAccess 는 해당 프로세스에 접근할 수 있는 권리를 의미한다. 핸들을 복제하기 위해서 PROCESS_DUP_HANDLE 옵션으로 프로세스에 접근한다. bInheritHandle 은 해당 프로세스를 이 프로세스에서 상속할 것인지를 지정한다. dwProcessId 는 열 프로세스의 식별자를 전달한다.

```
BOOL WINAPI DuplicateHandle(
  __in    HANDLE hSourceProcessHandle,
  __in    HANDLE hSourceHandle,
  __in    HANDLE hTargetProcessHandle,
  __out   LPHANDLE lpTargetHandle,
  __in    DWORD dwDesiredAccess,
  __in    BOOL bInheritHandle,
  __in    DWORD dwOptions
);
```

프로세스를 연 다음 해당 핸들을 복제하기 위해서 DuplicateHandle API 를 사용한다. hSourceProcessHandle 에 열린 복제할 프로세스 핸들과 hSourceHandle 에는 원 소스 핸들, hTargetProcessHandle 에는 복제될 핸들을 가질 프로세스 핸들, lpTargetHandle 에는 복제될 핸들을 가질 핸들 포인터를 전달한다. dwDesiredAccess 는 dwOptions 에 사용하는 인자가 DUPLICATE_SAME_ACCESS 일 경우에는 무시한다. bInheritHandle 는 핸들을 새 프로세스에서 상속할 것인지를 지정한다. dwOptions 는 원본 핸들과 같은 접근이 가능한 핸들로 복제하기 위해서 DUPLICATE_SAME_ACCESS 를 지정한다.

3.3 파일 정보 추출 과정

핸들의 복제가 성공하면 해당 오브젝트에 대해서 정보를 얻기 위해서 NtQueryObject API 를 이용한다.

```
NTSTATUS NtQueryObject(
  __in_opt    HANDLE Handle,
  __in                OBJECT_INFORMATION_CLASS
ObjectInformationClass,
  __out_opt   PVOID ObjectInformation,
  __in        ULONG ObjectInformationLength,
  __out_opt   PULONG ReturnLength
);
```

NtQueryObject 는 오브젝트의 정보를 가져올 수 있는 API 이다. NtQueryObject 를 이용하여 오브젝트의 기본 정보를 받아온다. Handle 에는 정보를 가져올 오브젝트의 핸들을 전달하고 ObjectInformationClass 에는 오브젝트의 정보를 수집하기 위해서 ObjectBasicInformation 값을 각각 전달한다. ObjectInformation 에는 PUBLIC_OBJECT_BASIC_INFORMATION 구조체를 넘겨서 값을 받아온다. ObjectInformationLength 에는 해당 구조체의 크기를 전달한다. ReturnLength 에는 전달받은 정보의 크기를 전달한다. 해당 오브젝트가 정상적으로 존재하는 오브젝트라면 NtQueryObject API 를 호출하여 ObjectNameInformation 을 전달하여 해당 오브젝트의 이름을 획득한다. 획득한 오브젝트의 이름이 “\Device\\Hard"로 시작하면 하드디스크의 파일을 의미하므로 해당 핸들 오브젝트를 프로세스에 연결시킨다.

4. 개발

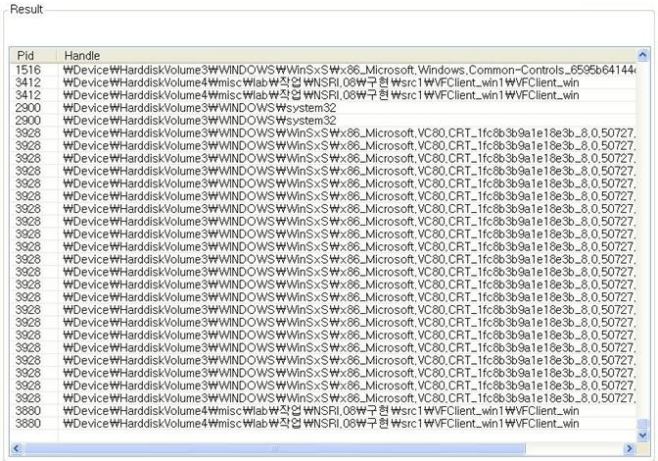
본 절에서는 동작 중인 메모리에서의 파일 정보 수집 도구의 개발에 관해 기술한다. 파일 정보 수집 도구는 대상 컴퓨터에서 파일 정보를 수집하는 클라이언트와 파일 정보를 관리하는 서버로 구성된다. 클라이언트에서 파일 정보를 수집한 후 네트워크를 통해 서버로 전송하면 서버는 파일 정보를 분석해 시스템에서 사용 중인 파일의 목록과, 파일을 사용하는 프

로세스의 ID 를 표시한다. 그림 4.1 은 3 절에서 기술한 파일 정보 수집 과정을 나타낸다.

```
getWinFileHandleList()
{
  LoadLibrary();
  GetProcAddress();
  EnablePrivilege();
  allocate a space for SYSTEM_PROCESS;
  pSysHandleInformation=NtQuerySystemInfomation();
  if(result == false)
  {
  get more space of SYSTEM_PROCESS;
  pSysHandleInformation=NtQuerySystemInfomation();
  }
  for(pSysHandleInformation->dwCount)
  {
    if(pSysHandleInformation->Handles->ObjectType == FILE)
    {
      OpenProcess();
      DuplicateHandle();
      NtQueryObject(ObjectBasicInformation);
      NtQueryObject(ObjectTypeInformation);
      NtQueryObject(ObjectNameInformation);
      runNtQueryInformationFile(FileNameInformation) with Thread;
    }
  }
}
```

<그림 4.1> 동작 중인 메모리에서 파일 정보 수집 과정

그림 4.2 는 관리 서버에서 나타낸 로드된 파일의 목록과 연관된 프로세스 ID 를 보여준다. 그림에서 왼오른쪽 부분에 Handle 로 표시된 부분이 시스템에서 현재 사용 중인 파일들을 목록이며, Pid 부분이 파일에 접근하고 있는 프로세스의 ID 를 나타낸다.



<그림 4.2> 동작 중인 메모리에서 파일 정보 수집 결과

5. 결론과 향후 연구 방향

본 논문에서는 메모리 포렌식 기법을 사용해 시스템에서 동작 중인 메모리에 존재하는 파일 정보를 수집하는 도구를 제안하였다. 수집 도구가 실행된 프로세스의 권한을 관리자 모드인 디버그 모드로 상승시키고, 해당 프로세스의 핸들을 복제하여 핸들 정보를 수집하였다. 그리고 수집한 핸들을 통해 시스템에서 사용 중인 파일에 대한 정보를 수집하였다. 윈도우의 Native API 를 통해 해당 과정을 수행하였고, 수집한 정보는 네트워크를 통해 외부의 관리 서버로 전송하여 일괄적으로 관리하는 방식을 사용했다.

동작 중인 메모리에서 파일 정보를 수집하는 과정에서 사용한 Native API 들이 악성 프로그램에 의해 변조되었을 경우 열린 파일 정보의 신뢰성을 검증할 수 없다는 단점이 존재한다. 또한, 메모리에서 수집할 수 있는 정보들이 디지털 포렌식에서 의미를 가지기 위해서는 네트워크로 연결된 외부 호스트와 프로세스 등 여러 가지 정보를 더 조합해야 한다. 향후에는 메모리 정보 수집을 위해 사용하는 시스템 기능들의 변조 여부 검증 방법과 메모리에서 수집한 정보들을 연관시켜 보여줄 수 있는 방법에 대한 연구가 진행되어야 한다.

참고문헌

[1] M. Burdach. “Digital forensics of the Physical Memory,” http://forensic.seccure.net/pdf/mburdach_digital_forensics_of_physical_memory.pdf

[2] E. Huebner, D. Bem, F. Henskens, M. Wallis, “Persistent Systems Techniques in Forensic Acquisition of Memory,” Digital Investigation, Vol. 4, No. 3-4, 2007

[3] G. Garcia, “Forensic Physical Memory Analysis: An Overview of Tools and Techniques,” TKK T-110,5290 Seminar on Network Security

[4] A. Schuster, “The Impact of Microsoft Windows Pool Allocation Strategies on Memory Forensics,” Digital Investigations Vol.5, 2008

[5] B. Dolan-Gavitt, “The VAD tree: A process-eye view of physical memory,” Digital Investigations, Vol.4, 2007

[6] B. Schatz, “BodySnatcher: Towards Reliable Volatile Memory Acquisition by Software,” Digital Investigation, Vol.4, 2007

[7] Win32dd, http://win32dd.msuiche.net

[8] Helix, www.e-fense.com/helix