Debugger

프로그램의 에러나 구조를 파악하기 위한 프로그램.

종류

White-box debugger

소스 코드를 확인할 수 있다. 대부분의 IDE가 제공하는 디버거가 이에 속한다.

Black-box debugger

소스 코드를 확인할 수 없으며 디어셈블된 결과만 확인 가능하다. Reversing Engineering할 때 쓰는 툴들이 대부분 이에 속한다. Black-box debugger는 프로그램 제어 권한 정도에 따라 다음 두 가지로 나뉜다.

User mode

최소한의 제어 권한을 가진다.

Kernel mode

운영 체제의 핵심 기능, 드라이버 등의 low-level component를 다룰 수 있다.

디버그 이벤트

디버거는 디버그 이벤트가 발생할 때까지 무한정 기다리게 된다. 대표적인 디버그 이벤트는 다음과 같다.

- Breakpoint hits
- Memory violations
- 디버깅한 프로그램에 의한 exceptions

디버그 이벤트가 발생하면 이에 대응되는 이벤트 핸들러가 호출된다.

Breakpoint

Breakpoint를 사용하여 프로세스를 정지시킬 수 있다. Breakpoint는 방법에 따라 크게 세 가지로 분류할 수 있다.

Software breakpoint

CPU가 실행하는 기계어 명령인 opcode(Operation Code)에 interrupt 3 (INT 3) instruction을 삽입하는 방식이다. 예를 들면 다음과 같다.

Breakpoint 설정 전 opcode

```
0x44332211: 8BC3 MOV EAX, EBX
```

Breakpoint 설정 후 opcode

```
0x44332211: CCC3 MOV EAX, EBX
```

단, 메모리에 로드된 실행 프로그램의 바이너리 내용을 변경하는 것이므로 CRC 체크섬을 사용하는 경우 이 값을 변경시킬 수 있다.

Hardware breakpoint

CPU의 debug register(DR0 ~ DR7)를 이용하는 breakpoint이다. 프로그램을 변경할 수 없으며 breakpoint가 적게 필요한 경우에 사용할 수 있다. Debug register의 각 역할은 다음과 같다.

Debug register	역할
DR0 ~ DR3	Breakpoint의 주소 저장
DR4, DR5	Reserved
DR6	디버그 이벤트의 type 결정
DR7	Breakpoint on/off, flag, length 결정

4개의 주소, 최대 4바이트에 대해서만 break point를 걸 수 있다는게 단점.

Memory breakpoint

메모리의 특정 영역(페이지)에 대한 권한을 변경하는 방법이다. 메모리 페이지의 권한에 대한 몇몇 예를 들면 다음과 같다.

권한	의미
Page execution	실행을 가능하게 한다. 프로세스가 read/write하려 하면 access violation을 발생시킨다.
Page read	Page에 대한 읽기만 가능하게 한다. Write/execution하는 경우 access violation을 발생시킨다.
Page write	Page에 대한 쓰기를 가능하게 한다.
Guard page	Page에 대한 어떤 접근도 못하게 한다. 접근하는 경우 예외를 발생시키고 page를 원래 상태로 돌린다.

위 권한을 조합하여 지정할 수도 있다. 위 권한 중 gaurd page가 reversing engineering에 유용하다.

관련 윈도우 API

프로세스

```
BOOL WINAPI CreateProcessA(
    LPCSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);
```

프로세스를 실행시킨다. lpApplicationName, lpCommandLine, dwCreationFlags, lpStartupInfo, and lpProcessInformation 정도가 중요한 파라메터이다. 그 외 파라메터는 보통 NULL로 주면 된다. STARTUPINFO, PROCESS_INFORMATION 구조체에 대한 내용은 MSDN을 참고한다.

CreateProcess : http://msdn.microsoft.com/en-us/library/ms682425.aspx
STARTUPINFO : http://msdn.microsoft.com/en-us/library/ms686331.aspx
PROCESS_INFORMATION : http://msdn.microsoft.com/en-us/library/ms686331.aspx

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

프로세스를 연 후 핸들을 리턴한다. 특정 프로세스의 디버깅을 위해 이 핸들을 사용하는 경우가 많다. 디버깅을 위해선 dwDesiredAccess는 PROCESS_ALL_ACCESS, bInheritHandle는 FALSE로 설정하면 된다. dwProcessId는 프로세스의 PID값을 넣어주면 된다. 이는 작업 관리자를 통해 확인할 수 있다.

OpenProcess : http://msdn.microsoft.com/en-us/library/ms684320.aspx

```
BOOL WINAPI DebugActiveProcess(
    DWORD dwProcessId
);
```

프로세스를 디버거에 attach한다. Attach가 되면 디버거가 디버그 이벤트를 제어할 수 있게 된다.

DebugActiveProcess : http://msdn.microsoft.com/en-us/library/ms679295.aspx

디버그

```
BOOL WINAPI WaitForDebugEvent(
```

```
    LPDEBUG_EVENT lpDebugEvent,
    DWORD dwMilliseconds
);
```

```
typedef struct DEBUG_EVENT {
    DWORD dwDebugEventCode;
    DWORD dwProcessId;
    DWORD dwThreadId;
    union {
        EXCEPTION_DEBUG_INFO Exception;
        CREATE_THREAD_DEBUG_INFO CreateThread;
        CREATE_PROCESS_DEBUG_INFO CreateProcessInfo;
        EXIT_THREAD_DEBUG_INFO ExitThread;
        EXIT_PROCESS_DEBUG_INFO ExitProcess;
        LOAD_DLL_DEBUG_INFO LoadDll;
        UNLOAD_DLL_DEBUG_INFO UnloadDll;
        OUTPUT_DEBUG_STRING_INFO DebugString;
        RIP_INFO RipInfo;
    }u;
};
```

디버그 이벤트를 정해진 시간만큼 기다린다. dwMilliseconds를 INFINITE로 설정하면 무한정 기다리게 된다. lpDebugEvent에 기다릴 이벤트에 대한 정보를 기록한다. 디버그 이벤트를 잡아내면 특정 이벤트 핸들러를 통해 원하는 동작을 할 수 있게 된다.

WaitForDebugEvent : http://msdn.microsoft.com/en-us/library/ms681423.aspx
DEBUG_EVENT : http://msdn.microsoft.com/en-us/library/ms679308.aspx

```
BOOL WINAPI ContinueDebugEvent(
    DWORD dwProcessId,
    DWORD dwThreadId,
    DWORD dwContinueStatus
);
```

디버그 이벤트에 대한 이벤트 핸들러 동작을 수행 후 프로세스를 다시 시작하기 위한 함수이다. 앞의 두 파라메터는 WaitForDebugEvent 함수가 리턴된 후 DEBUG_EVENT 구조체의 변수 중 dwProcessId와 dwThreadId가 초기화 되면 얻을 수 있다. dwContinueStatus는 DBG_CONTINUE와 DBG_EXCEPTION_NOT_HANDLED가 올 수 있으며 각각 프로세스 실행을 계속할지 exception을 실행할지를 의미한다.

ContinueDebugEvent : http://msdn.microsoft.com/en-us/library/ms679285.aspx

쓰레드

```
HANDLE WINAPI OpenThread(
    DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    DWORD dwThreadId
```

```
);
```

OpenThread : http://msdn.microsoft.com/en-us/library/ms684335.aspx

```
HANDLE WINAPI CreateToolhelp32Snapshot(
    DWORD dwFlags,
    DWORD th32ProcessID
);
```

CreateToolhelp32Snapshot : http://msdn.microsoft.com/en-us/library/ms682489.aspx

```
BOOL WINAPI Thread32First(
    HANDLE hSnapshot,
    LPTHREADENTRY32 lpte
);
```

```
typedef struct THREADENTRY32{
    DWORD dwSize;
    DWORD cntUsage;
    DWORD th32ThreadID;
    DWORD th32OwnerProcessID;
    LONG tpBasePri;
    LONG tpDeltaPri;
    DWORD dwFlags;
};
```

Thread32First : http://msdn.microsoft.com/en-us/library/ms686728.aspx
THREADENTRY32 : http://msdn.microsoft.com/en-us/library/ms686735.aspx

```
BOOL WINAPI GetThreadContext(
    HANDLE hThread,
    LPCONTEXT lpContext
);

BOOL WINAPI SetThreadContext(
    HANDLE hThread,
    LPCONTEXT lpContext
);
```

GetThreadContext : http://msdn.microsoft.com/en-us/library/ms679362.aspx
SetThreadContext : http://msdn.microsoft.com/en-us/library/ms680632.aspx

```
typedef struct CONTEXT {
    DWORD ContextFlags;
    DWORD Dr0;
    DWORD Dr1;
    DWORD Dr2;
    DWORD Dr3;
    DWORD Dr6;
    DWORD Dr7;
```

```
    FLOATING_SAVE_AREA FloatSave;
    DWORD SegGs;
    DWORD SegFs;
    DWORD SegEs;
    DWORD SegDs;
    DWORD Edi;
    DWORD Esi;
    DWORD Ebx;
    DWORD Edx;
    DWORD Ecx;
    DWORD Eax;
    DWORD Ebp;
    DWORD Eip;
    DWORD SegCs;
    DWORD EFlags;
    DWORD Esp;
    DWORD SegSs;
    BYTE ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];
};
```

CONTEXT : http://msdn.microsoft.com/en-us/library/windows/desktop/ms679284(v=vs.85).aspx

Breakpoint

Soft Breakpoints

```
BOOL WINAPI ReadProcessMemory(
    HANDLE hProcess,
    LPCVOID lpBaseAddress,
    LPVOID lpBuffer,
    SIZE_T nSize,
    SIZE_T* lpNumberOfBytesRead
);

BOOL WINAPI WriteProcessMemory(
    HANDLE hProcess,
    LPCVOID lpBaseAddress,
    LPCVOID lpBuffer,
    SIZE_T nSize,
    SIZE_T* lpNumberOfBytesWritten
);
```

ReadProcessMemory : http://msdn.microsoft.com/en-us/library/ms680553.aspx
WriteProcessMemory : http://msdn.microsoft.com/en-us/library/ms681674.aspx

```
FARPROC WINAPI GetProcAddress(
    HMODULE hModule,
    LPCSTR lpProcName
```

```
);

HMODULE WINAPI GetModuleHandle(
    LPCSTR lpModuleName
);
```

GetProcAddress : http://msdn.microsoft.com/en-us/library/ms683212.aspx
GetModuleHandle : http://msdn.microsoft.com/en-us/library/ms683199.aspx

Memory Breakpoints

```
void WINAPI GetSystemInfo(
    _Out_  LPSYSTEM_INFO lpSystemInfo
);

typedef struct _SYSTEM_INFO {
    union {
        DWORD  dwOemId;
        struct {
            WORD wProcessorArchitecture;
            WORD wReserved;
        };
    };
    DWORD     dwPageSize;
    LPVOID    lpMinimumApplicationAddress;
    LPVOID    lpMaximumApplicationAddress;
    DWORD_PTR dwActiveProcessorMask;
    DWORD     dwNumberOfProcessors;
    DWORD     dwProcessorType;
    DWORD     dwAllocationGranularity;
    WORD      wProcessorLevel;
    WORD      wProcessorRevision;
} SYSTEM_INFO;
```

GetSystemInfo : http://msdn.microsoft.com/en-us/library/ms724381.aspx
SYSTEM_INFO : http://msdn.microsoft.com/en-us/library/ms724958.aspx

```
SIZE_T WINAPI VirtualQuery(
    HANDLE hProcess,
    LPCVOID lpAddress,
    PMEMORY_BASIC_INFORMATION lpBuffer,
    SIZE_T dwLength
);

typedef struct MEMORY_BASIC_INFORMATION{
    PVOID BaseAddress;
    PVOID AllocationBase;
    DWORD AllocationProtect;
    SIZE_T RegionSize;
```

```
    DWORD State;
    DWORD Protect;
    DWORD Type;
}
```

VirtualQueryEx : http://msdn.microsoft.com/en-us/library/aa366907.aspx
MEMORY_BASIC_INFORMATION : http://msdn.microsoft.com/en-us/library/aa366775.aspx

```
BOOL WINAPI VirtualProtectEx(
    HANDLE hProcess,
    LPVOID lpAddress,
    SIZE_T dwSize,
    DWORD flNewProtect,
    PDWORD lpflOldProtect
);
```

VirtualProtectEx : http://msdn.microsoft.com/en-us/library/aa366899

From:
http://obg.co.kr/doku/ - **OBG WiKi**

Permanent link:
http://obg.co.kr/doku/doku.php?id=programming:reverse_engineering:debugger

Last update: **2020/11/29 14:09**