

# 윈도우 디버거

Python으로 살펴보는 윈도우 디버거 원리와 구현

## ○ 작성

- 2009년 10월 26일
- 양봉열 (xeraph@nchovy.com)

## ○ 목표

- 윈도우 디버거의 구조 이해

## ○ 라이선스

- 크리에이티브 커먼즈 저작자표시-비영리-변경금지 2.0 대한민국  
<http://creativecommons.org/licenses/by-nc-nd/2.0/kr/>

## ○ 참조

- [Gray Hat Python](#) by Justin Seitz
- 소스 다운로드: <http://nostarch.com/ghpython.htm>  
(Python 2.x 버전 기준)

# 디버거 기본 개념

## ○ 디버거의 역할

- 프로세스의 런타임 동작 추적 및 동적 분석
- 익스플로잇 개발, 퍼저 개발, 맬웨어 조사의 필수 요소

## ○ 디버거의 기능

- 프로세스를 실행시키거나, 정지시키거나, 한 단계씩 실행시킬 수 있는 기능
- 중단점 설정 기능
- 레지스터와 메모리 제어
- 예외를 잡을 수 있는 기능

## ○ 디버거의 분류

- 화이트박스 디버거: IDE에 통합된 디버거로 풍부한 정보를 활용할 수 있음
- 블랙박스 디버거: 리버스 엔지니어링이나 블랙박스 디버깅에 사용됨 (소스 없음)
  - 유저모드 디버거: Ring 3에서 동작하는 일반 응용프로그램 디버깅의 경우
  - 커널모드 디버거: 드라이버 디버깅 등 커널 수준의 디버깅을 요하는 경우

## ○ x86 범용 레지스터

- EAX: 누산 레지스터, 대부분의 계산 명령이 EAX를 사용하도록 최적화됨
- EDX: 데이터 레지스터, EAX와 함께 계산할 때 주로 사용됨
- ECX: 카운터 레지스터, 루프 돌 때 사용됨
- ESI: 데이터 원본을 가리키는 레지스터
- EDI: 데이터 사본을 가리키는 레지스터
- EBP: 베이스 포인터 레지스터
- ESP: 스택 포인터 레지스터
- EBX: 일반 데이터 레지스터
- EIP: 현재 실행되고 있는 명령의 주소를 가리키는 레지스터

## ○ C 함수 호출

– int my\_socks(color\_one, color\_two, color\_three);

## ○ x86 어셈블리 변환

– push color\_three  
push color\_two  
push color\_one  
call my\_socks



## ○ 흐름

- 디버깅 이벤트가 발생할 때까지 무한 대기
- 이벤트가 발생하면 디버거는 정지하고 사용자의 명령을 대기하게 됨
  - 중단점을 만나는 경우
  - 메모리 위반이 발생한 경우
  - 디버깅 대상 프로그램에서 예외를 던진 경우

## ○ 이벤트 코드 분류

1. EXCEPTION\_DEBUG\_EVENT: u.Exception
2. CREATE\_THREAD\_DEBUG\_EVENT: u.CreateThread
3. CREATE\_PROCESS\_DEBUG\_EVENT: u.CreateProcessInfo
4. EXIT\_THREAD\_DEBUG\_EVENT: u.ExitThread
5. EXIT\_PROCESS\_DEBUG\_EVENT: u.ExitProcess
6. LOAD\_DLL\_DEBUG\_EVENT: u.LoadDll
7. UNLOAD\_DLL\_DEBUG\_EVENT: u.UnloadDll
8. OUTPUT\_DEBUG\_STRING\_EVENT: u.DebugString
9. RIP\_EVENT: u.RipInfo

## ○ 소프트 중단점

### - 원리

- INT3 명령어(0xCC)를 실행하는 경우 CPU가 멈추게 됨
- 소프트 중단점을 걸려면 원래의 명령어를 덮어써야 함
- 디버거 프로세스를 멈추고 디버거의 중단점 예외 처리기로 제어를 넘기게 됨

### - 흐름

- 특정 주소에 소프트 중단점을 걸면서 원래 바이트를 백업함
- 해당 주소에 0xCC를 덮어쓰기
- CPU가 0xCC를 실행하면서 INT3 인터럽트 발생
- 디버거는 EIP를 살펴보고 바이트 값을 복원한 후 실행 재개

### - 단점

- 메모리의 실행 이미지를 덮어쓰게 되므로 체크섬이 달라지게 됨
- 체크섬을 검사하여 디버거의 존재 유무를 알아채고 분석을 방해할 수 있음

## ○ 하드웨어 중단점

### - 원리

- CPU 수준에서 설정 가능한 디버그 레지스터를 이용, INT1 발생
- DR0 ~ DR3: 중단점 주소 저장
- DR4 ~ DR5: 예약됨
- DR6: 중단점 상태
- DR7: 하드웨어 중단점 활성화 스위치 역할 및 중단점 조건 저장
  - 특정 주소의 명령이 실행될 때 중단
  - 특정 주소에 데이터가 쓰여질 때 중단
  - 특정 주소에 읽고 쓸 때 중단

### - 흐름

- CPU가 명령을 실행할 때마다 하드웨어 중단점 조건을 검사함

### - 단점

- 한 번에 4개 밖에 못 씀

## ○ 메모리 중단점

### – 원리

- 실제 중단점은 아니지만 페이지 퍼미션을 변경해서 유사한 효과를 낼 수 있음
  - 각각 실행, 읽기, 쓰기 권한을 위반하는 경우 예외 발생
  - 가드 페이지: 해당 페이지에 접근 시 한 번 예외를 내고 원래 상태로 되돌아감
    - » 특정 메모리 영역의 변화를 감지하려는 경우 유용하게 사용됨

# 윈도우 디버거 개발

디버깅 API와 Python을 이용한 실습

## ○ 디버거 연결 방식

- 프로세스 생성 시 연결: 프로세스 동작을 처음부터 완전히 제어 가능
  - [CreateProcess\(\)](#): dwCreationFlags 매개변수에 DEBUG\_PROCESS 설정
- 기존 프로세스에 연결
  - [OpenProcess\(\)](#): dwDesiredAccess 매개변수에 PROCESS\_ALL\_ACCESS 설정

## ○ 디버거 연결 및 이벤트 처리

- 디버거 연결 및 해제
  - [DebugActiveProcess\(\)](#): 프로세스에 디버거 부착
  - [DebugActiveProcessStop\(\)](#): 프로세스에서 디버거 탈착
- 이벤트 처리
  - [WaitForDebugEvent\(\)](#): 지정된 시간 동안 디버그 이벤트 대기
  - [ContinueDebugEvent\(\)](#): 디버그 이벤트 처리 후 실행 재개 요청

- 스레드 및 실행문맥 제어: CPU 실행 문맥은 스레드별로 유지됨
  - OpenThread()
    - 스레드 ID를 이용하여 스레드 핸들 획득
  - CreateToolhelp32Snapshot()
    - 프로세스 내의 힙, 모듈, 스레드 정보 획득
  - Thread32First() / Thread32Next()
    - 스레드 정보를 THREADENTRY 구조체로 가져옴
  - GetThreadContext()
    - CPU 레지스터 상태와 플래그 등을 포함한 실행 문맥을 조회
  - SetThreadContext()
    - CPU 레지스터나 플래그 등 원하는대로 CPU 실행 문맥 재설정
    - 반드시 스레드가 멈춘 상태에서 호출해야 함
  - SuspendThread() / ResumeThread()
    - 스레드 일시 정지 및 실행 재개

## ○ 소프트 중단점

- ReadProcessMemory()
  - 프로세스 실행 이미지에서 해당 주소의 바이트를 백업할 때 사용
- WriteProcessMemory()
  - 프로세스 실행 이미지에 0xCC를 쓰거나 원래 바이트 값 복원할 때 사용

## ○ 하드웨어 중단점

- SetThreadContext()
  - 디버그 레지스터 설정

## ○ 메모리 중단점

- GetSystemInfo()
  - 페이지 크기 조회
- VirtualQueryEx()
  - MEMORY\_BASIC\_INFORMATION 구조체로 특정 주소가 속한 페이지 정보 조회
- VirtualProtectEx()
  - MEMORY\_BASIC\_INFORMATION의 Protect 멤버에 PAGE\_GUARD를 설정