파일 퍼저 UFuz3 분석

이용일(foryou2008@nate.com)

2008-07-26

0. 개요

UFuz3는 eEye에서 공개한 파일 퍼저입니다. UFuz3의 기능 분석 및 상세 분석을 통해, 퍼저는 어떤 모습을 갖추고 있는지, 또한 퍼저는 내부적으로 어떤 방식으로 동작하는지 살펴보려고 합니다.

1. 기능 분석

UFuz3는 파일을 파싱하는 루틴 내에 Integer Overflow가 존재하는지를 검사합니다. 따라서 퍼징의 대상이 되는 대상 프로그램은 데이터 파일을 입력으로 받아야 하는 제한이 있습니다.

다음은 파일 파싱 루틴 내에 Integer Overflow가 존재하는 예제입니다. (UFuz3 도움말에 있는 예제 소스입니다. 정확히 말해, 이 소스는 Integer Overflow를 포함하고 있다라고 볼 수 없습니다.)

```
#include <windows.h>
#include <stdio.h>

void subfunc(char *buf, DWORD len)
{
	FILE	*fp;
	char	*mem;

	mem = malloc(32768);
	memcpy(mem, buf, len); // buffer overflow
	if ((fp = fopen(mem, "r")))
		fclose(fp);
	free(mem);
```

```
}
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                      LPTSTR lpCmdLine, int nCmdShow)
{
        FILE    *fp;
        char    buf[64000];
        char    szFileName[512];
        DWORD   v;
        DWORD   dwArgc;
        WCHAR   **lpszArgv;
        lpszArgv = CommandLineToArgvW(GetCommandLineW(),&(dwArgc));

        if (dwArgc==2){
                memset(szFileName,0,sizeof(szFileName));
                _snprintf(szFileName,sizeof(szFileName)-1,"%ws",lpszArgv[1]);

                if ((fp=fopen(szFileName,"rb"))){
                        fread(buf, 1, sizeof(buf), fp);
                        v=*((DWORD *)(buf+16)); // 파일 시작 + 오프셋 0x10 위치의 값을
                                                // subfunc의 두번째 인자인 length로 사용한다.
                        subfunc(buf+4,v);
                        fclose(fp);
                }else{
                        sprintf(buf,"Can not open '%s'",szFileName);
                        MessageBox(NULL,buf,"Error",MB_OK);
                }
        }else{
                sprintf(buf,"Invalud number of arguments (%d)",dwArgc);
                MessageBox(NULL,buf,"Error",MB_OK);
        }
        return 0;
}
```

[소스 1] UFuz3 도움말 예제 소스

[소스 1]은 특정 파일을 오픈하고, 오픈한 파일의 시작으로부터 0x10 위치에 있는 더블 워드 값을

length로 활용합니다. 이 때 length의 바운더리 체크를 하지 않아, 할당된 메모리의 크기보다 더 많은 length 만큼의 데이터를 쓸 수 있는 기회가 발생합니다.

만약 데이터 파일이 다음과 같이 조작되었다고 가정해 봅시다.

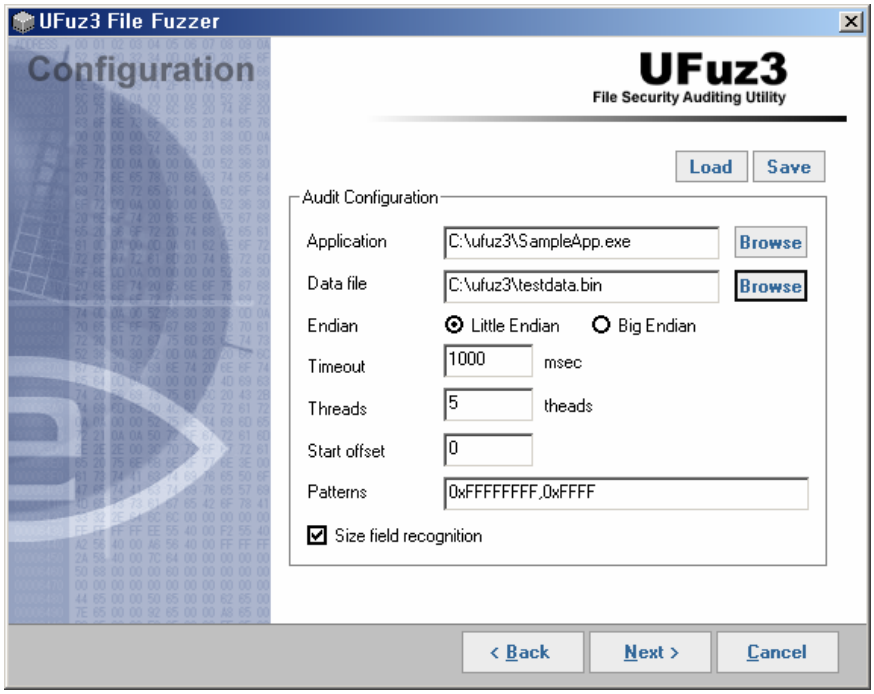
```
ADDRESS    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F    0123456789ABCDEF
-------------------------------------------------------------------------------
 00000000   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
 00000010   00 90 00 00 61 61 61 61 61 61 61 61 61 61 61 61    °...aaaaaaaaaaaa
 00000020   61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61    aaaaaaaaaaaaaaaa
```

데이터 파일의 0x10 – 0x13 까지는 length로 활용되는 부분인데, 0x00009000로 조작되었습니다. 따라서 32768바이트 크기의 메모리에 36864 바이트만큼의 파일 데이터를 쓰게 될 것이므로 오버플로우가 발생할 것입니다.

* [소스 1]은 UFuz3의 도움말에 들어가 있는 예제 소스인데, Integer Overflow의 예제로 사용하기에는 적절하지 않다고 생각합니다. Integer Overflow라는 것은 말 그대로, Integer Variable이 수용할 수 있는 값의 범위를 초과하게 되어 예상하지 못한 결과를 만들어내게 되는 것인데, Widthness overflow나 Arithmetic Operation의 결과로써 그 원인이 발생합니다. 하지만 [소스 1]의 경우에는, 어떤 Arithmetic Operation의 결과로서 Integer Variable에 오버플로우가 발생하는 것이 아니라, 단순히 Boundary Checking을 하지 않아서 발생하는 오버플로우라 생각되어질 수 밖에 없습니다. 하지만 예제가 부적절하다고 해서 UFuz3가 Integer Overflow를 찾지 못한다는 것은 아닙니다. UFuz3는 데이터 파일의 여러 랜덤한 위치에 Integer Overflow를 일으킬만한 값을 적고, 대상 프로그램을 실행하여 Integer Overflow를 유도하고 있습니다.

이제, 실제 [소스 1]을 컴파일한 대상 프로그램과 데이터 파일을 가지고 UFuz3를 실행해보겠습니다.

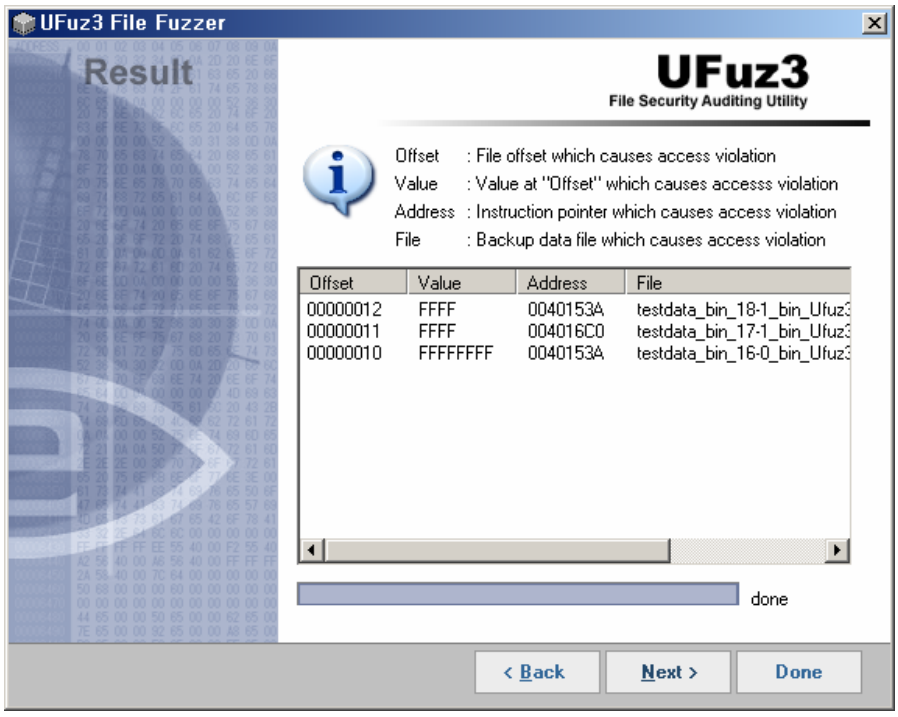
[그림 1] Ufuz3 GUI 실행 첫 화면



[그림 2] 퍼징 옵션 설정 화면

다음은 옵션 설명입니다.

옵션 이름	설명
Target Application	대상 프로그램을 지정합니다.
Data File	대상 프로그램이 사용할 데이터 파일을 지정합니다.
Endian	워드 인코딩 방식을 지정합니다. 보통 대부분의 윈도우 어플리케이션은 Little-Endian 방식을 사용합니다. 하지만 Real-Player와 같은 몇몇 어플리케이션은 Big-Endian 방식을 사용합니다.
Timeout	만약에 어플리케이션이 Exception을 발생하지 않을 경우에, 강제로 어플리케이션을 종료시켜야 할 것입니다. 지정한 Timeout 내에 어플리케이션이 종료하지 않으면, 강제로 종료합니다.
Threads	Ufuz3은 멀티스레딩을 지원합니다. Threads에서 지정한 수만큼 스레드가 생성되어, 분산적으로 데이터 파일을 조작하여, 대상 프로그램을 검사합니다. Threads 옵션을 잘 활용하면, 좀 더 빠르게 결과를 도출해낼 수 있습니다.
Start Offset	데이터 파일의 시작 위치를 지정합니다.
Patterns	데이터 파일에 쓰여질 패턴을 지정합니다. ","를 사용해 여러 개의 패턴도 지정 가능합니다.
Size Field Recognition	Length 필드라고 판단되지 않는 데이터 파일 영역은 skip합니다. 보통 length 필드는 "32 1E 00 00" 과 같이 0x00을 포함합니다. 따라서 만약 이 옵션을 켜 놓았다면, 데이터 파일에서 0x00을 포함하지 않는 필드는 Length 필드가 아니라고 판단하여 Skip 할 것입니다.



[그림 3] Ufuz3의 퍼징 결과

[그림 3]의 결과를 해석해보겠습니다.

Offset	Value	Address	설명
00000012	FFFF	0040153A	데이터 파일 오프셋 0x12 위치에 FFFF값을 적고, 대상 프로그램을 실행했더니, 0x0040153A 위치에서 예외가 발생했다는 의미입니다.
00000011	FFFF	004016C0	데이터 파일 오프셋 0x11 위치에 FFFF값을 적고, 대상 프로그램을 실행했더니, 0x004016C0 위치에서 예외가 발생했다는 의미입니다.
00000010	FFFFFFFF	0040153A	데이터 파일 오프셋 0x10 위치에 FFFFFFFF값을 적고, 대상 프로그램을 실행했더니, 0x0040153A 위치에서 예외가 발생했다는 의미입니다.

2. 상세 분석

MS Windows에서는 프로그램을 디버깅 하기 위한 Debugger API Set을 제공합니다. UFuz3 역시 이러한 Debugger API를 사용하여 대상 프로그램을 실행하고, 예외가 발생하는지를 확인하고 있습니다.

UFuz3은 대상 프로그램을 디버그하기 위해 CreateProcess를 사용해 대상 프로그램을 실행합니다. 여기서 관심을 가져야 할 파라미터는 바로 프로세스의 생성 옵션을 지정하는 dwCreationFlags 부분입니다. dwCreationFlags 중에서 DEBUG_PROCESS 플래그 설명을 살펴봅시다. (From MSDN)

The calling thread starts and debugs the new process and all child processes created by the new process. It can receive all related debug events using the **WaitForDebugEvent** function. A process that uses DEBUG_PROCESS becomes the root of a debugging chain. This continues until another process in the chain is created with DEBUG_PROCESS. If this flag is combined with DEBUG_ONLY_THIS_PROCESS, the caller debugs only the new process, not any child processes. "CreateProcess를 호출한 스레드는 생성된 프로세스와 그 자식 프로세스까지 모두 디버그합니다. WaitForDebugEvent 함수를 사용해 디버그와 관련된 모든 이벤트를 받을 수 있습니다. … DEBUG_ONLY_THIS_PROCESS와 같이 사용되면, 자식 프로세스는 제외하고 오직 생성된 프로세스만 디버그합니다."

UFuz3는 아래와 같이 DEBUG_PROCESS 플래그를 사용하여 CreateProcess를 호출하는 것을 알 수 있습니다.

```
.text:00401241    push    ecx                 ; lpProcessInformation
.text:00401242    lea     edx, [esp+1D0h+StartupInfo]
.text:00401249    push    edx                 ; lpStartupInfo
.text:0040124A    push    eax                 ; lpCurrentDirectory
.text:0040124B    push    eax                 ; lpEnvironment
.text:0040124C    push    11h                 ; dwCreationFlags
.text:0040124E    push    1                   ; bInheritHandles
.text:00401250    push    eax                 ; lpThreadAttributes
```

```
.text:00401251                  push    eax                     ; lpProcessAttributes
.text:00401252                  push    esi                     ; lpCommandLine
.text:00401253                  push    eax                     ; lpApplicationName
.text:00401254                  mov     [esp+1F4h+lParam], eax
.text:00401258                  mov     [esp+3Ch], eax
.text:0040125C                  mov     [esp+40h], eax
.text:00401260                  mov     [esp+44h], eax
.text:00401264                  mov     [esp+1F4h+StartupInfo.cb], 44h
.text:0040126F                  call    ds:CreateProcessA
```

앞서 언급했듯이, 디버그 이벤트를 통지받기 위해, WaitForDebugEvent API를 사용한다고 하였습니다. UFuz3도 WaitForDebugEvent API를 사용할 것입니다. 한번 살펴봅시다.

```
.text:00401282                  mov     ecx, dwMilliseconds
.text:00401288                  mov     esi, ds:WaitForDebugEvent
.text:0040128E                  add     esp, 0Ch
.text:00401291                  push    ecx                     ; dwMilliseconds
.text:00401292                  lea     edx, [esp+1D0h+dwProcessId]
.text:00401296                  push    edx                     ; lpDebugEvent
.text:00401297                  call    esi ; WaitForDebugEvent
```

WaitForDebugEvent의 프로토타입은 다음과 같습니다. 즉, 지정한 dwMilliseconds만큼 디버그 이벤트를 기다리는 동안에 디버그 이벤트가 발생하면, lpDebugEvent에 디버그 이벤트 관련 정보를 채우고, 리턴하는 것입니다. 만약에 지정한 dwMilliseconds 내에 디버그 이벤트가 발생하지 않았거나, 예기치 못한 에러가 발생했다면 WaitForDebugEvent는 zero를 리턴하고, 지정한 시간 내에 디버그 이벤트가 발생했다면 nonzero를 리턴합니다.

```
BOOL WINAPI WaitForDebugEvent(
  __out   LPDEBUG_EVENT lpDebugEvent,
  __in    DWORD dwMilliseconds);
```

여기서 추론해볼 수 있는 부분은,

1) dwMilliseconds 부분에는 우리가 UFuz3 실행시에 옵션으로 지정했던 Timeout 값이 들어갈 것이다.

2) 만약 WaitForDebugEvent가 zero를 리턴했다면, 지정한 Timeout동안에 아무런 이벤트가 발생하지 않았다는 의미이며, 이는 곧 예외가 발생하지 않았음을 의미한다. 따라서 프로세스를 강제로 종료해야 할 것이다.

3) 만약 WaitForDebugEvent가 nonzero를 리턴했다면, 지정한 Timeout동안에 디버그 이벤트가 발생했다는 것을 의미한다. 따라서 디버그 이벤트의 내용을 살펴, 그것이 예외와 관련된 이벤트였다면, 대상 프로그램에서 예외가 발생했다는 것을 의미하며 이것은 UFuz3가 원하던 결과이다. 따라서 이러한 결과를 저장한다.

우선 1) 부터 확인해보겠습니다.

UFuz3의 퍼징 옵션 중 Timeout 값을 1000msec로 지정하고, 퍼징을 실행한 후 디버거를 통해 WaitForDebugEvent의 파라미터를 확인해봅니다. 다음은 ollydbg에서 확인한 모습입니다. 추론한 대로 우리가 지정한 Timeout 퍼징 옵션이 그대로 WaitForDebugEvent에서 사용되는 것을 알 수 있습니다.

```
00401291 . 51          PUSH ECX                          Timeout => 1000. ms
00401292 . 8D5424 24   LEA EDX,DWORD PTR SS:[ESP+24]
00401296 . 52          PUSH EDX                          pDebugEvent
00401297 . FFD6        CALL ESI                          WaitForDebugEvent
```

다음은 3)을 확인해 보겠습니다. WaitForDebugEvent에서 nonzero를 리턴했다면, WaitForDebugEvent의 첫번째 파라미터(DEBUG_EVENT 구조체 타입)는 유효한 값이 들어가 있을 것입니다.

다음은 DEBUG_EVENT 구조체의 정의입니다.

```
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;
} DEBUG_EVENT,   *LPDEBUG_EVENT;
```

DEBUG_EVENT의 dwDebugEventCode 멤버는 이름에서 알 수 있듯이, 디버그 이벤트가 무엇인지를 알려주는 ID입니다. UFuz3가 관심을 가지는 디버그 이벤트는 예외가 발생한 경우겠죠. 따라서 UFuz3는 dwDebugEventCode이 EXCEPTION 이벤트를 가리키는지를 살펴볼 것입니다.

다음 코드는 dwDebugEventCode가 EXCEPTION_DEBUG_EVENT(1)인지를 확인하는 부분입니다.

```
.text:004012A3 loc_4012A3:                                ; CODE XREF: sub_401200+100├j
.text:004012A3                  mov     eax, [esp+1CCh+dwProcessId]
;
; switch (dwDebugEventCode)
; {
; case EXCEPTION_DEBUG_EVENT:
;       goto loc_4012CD;
.text:004012A7                  sub     eax, 1  ; #define EXCEPTION_DEBUG_EVENT 1
.text:004012AA                  jz      short loc_4012CD
```

dwDebugEventCode에 따라 DEBUG_EVENT 구조체에서 참조해야 하는 멤버가 달라지게 됩니다. dwDebugEventCode가 EXCEPTION_DEBUG_EVENT일 경우에는, DEBUG_EVENT 구조체의 EXCEPTION_DEBUG_INFO Exception을 참조해야 합니다. EXCEPTION_DEBUG_INFO 구조체는 다음과 같습니다.

```
typedef struct _EXCEPTION_DEBUG_INFO
{
```

```
EXCEPTION_RECORD ExceptionRecord;
DWORD dwFirstChance;
} EXCEPTION_DEBUG_INFO,
```

EXCEPTION 이벤트도 여러가지 종류가 있을 것입니다. 잘못된 메모리 참조 예외(Access Violation)이나 Breakpoint 예외, Divide By Zero 예외 등이 있습니다. 이 중에서 UFuz3는 잘못된 메모리 참조 예외에 대해서만 캐치를 하는군요.

EXCEPTION 이벤트의 상세 종류는 EXCEPTION_RECORD의 ExceptionCode에서 알려주고 있습니다. UFuz3는 EXCEPTION_RECORD의 ExceptionCode 멤버값이 EXCEPTION_ACCESS_VIOLATION인지를 확인하여 디버그 프로세스가 Access Violation 예외가 발생하였는지를 판단하고 있습니다.

```
.text:004012CD loc_4012CD:                              ; CODE XREF: sub_401200+AA↑j
;
; switch (DEBUG_EVENT->Exception.ExceptionRecord.ExceptionCode)
; {
; case EXCEPTION_ACCESS_VIOLATION: // 0C0000005h
;       goto loc_401362;
.text:004012CD                  cmp     dword ptr [esp+2Ch], 0C0000005h
.text:004012D5                  jz      loc_401362
...
...
.text:00401362 loc_401362:                              ; CODE XREF: sub_401200+D5↑j
.text:00401362                  mov     eax, dword ptr [esp+1CCh+arg_4]
.text:00401369                  push    eax             ; char
.text:0040136A                  push    offset aException_acce ; "EXCEPTION_ACCESS_VIOLATION
(%d)₩n"
.text:0040136F                  call    sub_401000
.text:00401374                  mov     ecx, [esp+84h]
.text:0040137B                  push    ecx             ; char
.text:0040137C                  push    offset aDwfirstchanceD ; " dwFirstChance = %d₩n"
.text:00401381                  call    sub_401000
.text:00401386                  mov     edx, [esp+3Ch]
.text:0040138A                  push    edx             ; char
```

```
.text:0040138B    push    offset aExceptioncodeX ; " ExceptionCode = %x₩n"
.text:00401390    call    sub_401000
.text:00401395    mov     eax, [esp+50h]
.text:00401399    push    eax                     ; char
.text:0040139A    push    offset aExceptionaddrX ; " ExceptionAddr = %x₩n"
.text:0040139F    call    sub_401000
.text:004013A4    lea     ecx, [esp+1ECh+NewFileName]
.text:004013AB    push    ecx
.text:004013AC    push    offset ExistingFileName
.text:004013B1    mov     edi, 5
.text:004013B6    call    sub_4010E0
```

* Windows Debugger API Set에 대한 더 자세한 정보는 msdn을 참고해주세요.

3. 결론

UFuz3는 아주 기본적인 파일 퍼저입니다. 다른 퍼저에 비해 어떤 특징적인 부분을 제공하고 있지는 않습니다. 버그도 많이 보이고, 인터페이스도 사실 엉망입니다. 그리고 퍼징 결과 또한 만족스럽지 못하고요. eEye에서 저런 프로그램을 공개했다는 것이 다소 의문입니다(개인적인 견해입니다.) 하지만 파일 퍼징을 공부하는 입장에서, UFuz3을 사용해봄으로써 기본적으로 파일 퍼저가 어떤 기능을 수행하며, 또한 윈도우에서는 어떤 식으로 구현되는지를 알아볼 수 있었다는 데 의의를 두고 싶습니다.

파일 이름: UFuz3 분석
폴더: C:₩Documents and Settings₩hp₩바탕 화면
서식 파일: C:₩Documents and Settings₩hp₩Application Data₩Microsoft₩Templates₩Normal.dot
제목: UFuz3 분석
주제:
만든 이: m
키워드:
메모:
만든 날짜: 2008-07-26 AM 11:31:00
수정 횟수: 151
마지막으로 저장한 날짜: 2008-07-26 PM 2:49:00
마지막으로 저장한 사람: m
전체 편집 시간: 198 분
마지막으로 인쇄한 날짜: 2008-07-26 PM 2:49:00
문서량
페이지 수: 12
단어 수: 1,835 (약)
문자 수: 10,463 (약)