闲来逛逛 milw0rm.
找到一个 Cain Overflow Exploit.
见 http://www.milw0rm.com/exploits/7309
为熟练自己的调试技巧特来**分析**以下。
针对 Cain & Abel <= v4.9.24(其他的我没式，毕竟我只是**分析漏洞**而不只找**漏洞**)

作者使用的系统为 SP3，所以在 PERL 脚本中看到。 XP SP3: 0x7c9d30d7 jmp esp
如果你知道著名的中文 Win2000、XP、Win2003 的 JMP ESP 通用跳转地址，那么你一定会把他改成 7ffa4512。

击 CAIN 点击工具栏上"Remote Pass**word** Decoder Dialog" 按钮。打开生成的 s.rdp，一个计算器弹了出来。

好了我们现在来调试。
为了不让一调试就执行 SHELLCODE，我们把 SHELLCODE 开头加几个 CC(INT3 断点)。
载入 0D，不**分析**，F9 运行，一个异常，SHIFT+F9 过。同理点按钮.又一个异常，SHIFT+F9 过。打开 S.RDP。
准确中断到 CC 处.
已经执行到 SHELLCODE 了，现在我们需要知道到底是从哪来的。
查看堆栈。全覆盖成了 414141，而且再往下拉一点就是当前的 EIP。
可见，溢出原理是堆栈被某函数覆盖，然后跳到堆栈执行堆栈内的 SHELLCODE。
所以，我们考虑对 0012eed4 下内存写入断点(这里被覆盖成 7ffa4512 后面是 4141...+shellcode).

重新载入。再打开 s.rdp 前，停下。乘此机会，我们下 0012eed4 的内存写入断点.再按 shift+F9 断了下来。
接下来单步的走饼时刻留意内存数据窗口 0012eed4 的变化。没走几下，
00474EDD E8 DA821400 call 005BD1BC
00474EE2 83C4 18 add esp, 18
00474EE5 85C0 test eax, eax
00474EE7 0F84 81010000 je 0047506E

内存被刷的一下 覆盖了。看来，问题就在这个 CALL 里。

再来过。
这次我们 F7 到 CALL 里。
可以看堆栈内的参数

2000h ；十进制 8192 缓冲区长度
0012cec8 ；要写入的地址

稍微跟踪一下子程序。我们发现主要是一个循环
005BD1D9 FF4D 0C dec d**word** ptr [ebp+C] ；循环开始
005BD1DC 74 28 je short 005BD206
005BD1DE FF75 10 push d**word** ptr [ebp+10]
005BD1E1 E8 DA930000 call 005C65C0
005BD1E6 0FB7C0 movzx eax, ax
005BD1E9 3D FFFF0000 cmp eax, 0FFFF
005BD1EE 59 pop ecx

```
005BD1EF   74 0D          je      short 005BD1FE
005BD1F1   66:8906        mov     word ptr [esi], ax          ; 这里开始写入缓冲区
005BD1F4   46             inc     esi
005BD1F5   46             inc     esi
005BD1F6   66:3D 0A00     cmp     ax, 0A
005BD1FA   74 0A          je      short 005BD206
005BD1FC  ^ EB DB         jmp     short 005BD1D9
```

所以我们仔细看这段循环。
它不断的把 ax 里的数据写入地址 0012cec8 之后.长度是 2000h（8192）

我们关心的 0012eed4，因为他将被改写为 jmp esp 的地址。
未改写前
0012EED4 00474DDD   返回到 Cain.00474DDD 来自 Cain.00474E30
所以改写后子程序返回时候继续 jmp esp
于是执行该地址后面的 shellcode 了。

现在 exploit 的流程非常清楚了，ax 中的数据哪里来呢？

```
005BD1DE   FF75 10        push    dword ptr [ebp+10]
005BD1E1   E8 DA930000    call    005C65C0
```

这个 call 通过[ebp+10]01762f38 获得数据，返回到 eax 里。
所以我门要跟进去看看到底怎么来的数据，为什么会允许过长的数据，覆盖缓冲区

稍微读一下发现第一次的 call 把 rdp 的数据读到了 eax
文件读到
01764fa8---------01765f98
长度限制为 1000 且全是 AAA。。饼不能造成溢出。
跟踪发现，这个 call 后来又会再次读饼更改读入缓冲区的内容。
有兴趣的细细跟一下吧。
因此:AAAAA--->7ffa4512+AAAA+shellcode+AAAA。
此后循环继续覆盖缓冲区，最终覆盖了返回地址 0012eed4。产生溢出。