第一章 概述

1. 网络攻击：

   指利用安全缺陷或不当配置对网络信息系统的硬件、软件或通信协议进行攻击，损害网络信息系统的完整性、可用性、机密性和抗抵赖性，导致被攻击信息系统敏感信息泄露、非授权访问、服务质量下降等后果的攻击行为。

2. 网络：

   狭义上讲是计算机网络，广义上讲则是网络空间

3. 网络安全威胁：

   1) 协议缺陷
   2) 软件漏洞
   3) 策略弱点
   4) 恶意利用
   5) 硬件漏洞
   6) 管理不当

4. 网络攻击分类

   按照攻击者与被攻击者的物理位置进行分类

   1) 物理攻击
   2) 主动攻击
   3) 被动攻击
   4) 中间人攻击

5. 网络攻击步骤

   1) 信息收集
   2) 权限获取
   3) 安装后门
   4) 扩大影响
   5) 清除痕迹

第二章 信息收集

1. 信息收集：

   黑客为了更加有效地实施攻击而在攻击过程中对目标的所有探测活动

2. 内容

   - 域名、IP 地址
   - 防火墙、入侵检测等安全防范措施
   - 内部网络结构、域组织、用户电子邮件
   - 操作系统类型
   - 端口

- 系统架构
- 敏感文件或目录
- 应用程序类型

3. 分类

1) 主动：通过直接访问、扫描网站，这种将流量流经网站的行为
2) 被动：利用第三方的服务对目标进行访问了解，比例：Google 搜索

4. 公开信息收集

1) 利用 Web 服务（社会工程学）
2) 利用搜索引擎服务（google hacking）

•基本语法

And	与
OR	或
+	强制包含搜索项
-	非，去掉搜索项
“”	包含一个完整的语义
.	单个通配符
*	任意通配符

•高级操作符

site:	搜索具体服务器或域名的网页
filetype:	搜索特定类型的文件
intitle:	搜索网页标题
inurl:	搜索URL
intext:	搜索正文
link:	搜索连接到指定网页的网页

3) Whois 服务：查询已注册域名的拥有者信息
4) DNS 域名服务：如果 DNS 配置不当，可能造成内部主机名和 IP 地址对的泄漏 Nslookup 可查到服务器地址和 IP 地址以及域名服务器的传输内容

5. 网络扫描

1) 主机扫描
   a) 使用 ICMP 扫描——ping：向目标主机发送 ICMP Echo Request (type 8)数据包，等待回复的 ICMP Echo Reply 包(type 0)。
   b) 高级 IP 扫描技术：
      - 构造异常的 IP 包头

        目标主机会反馈 ICMP Parameter Problem Error 信息。常见的伪造错误字段为 Header Length Field 和 IP Options Field。
      - 在 IP 头中设置无效的字段值

        如果目标主机存活，反馈 ICMP Destination Unreachable 信息。
      - 错误的数据分片

        在规定的时间间隔内得不到更正时，目标主机将丢弃这些数据包，并向发送主机反馈 ICMP Fragment Reassembly Time Exceeded 报文。
      - 通过超长包探测内部路由器

        若构造得数据包长度超过目标系统所在路由器的 PMTU,且设置禁止分片标志，该路由器会反馈 Fragmentation Needed and Don’t Fragment Bit was Set 差错报文。
      - 反向映射探测

构造可能的内部 IP 地址列表，并向这些地址发送数据包。内部网络路由器接收到数据包时，会进行 IP 识别并路由，对不在服务范围的 IP 包发送 ICMP Host Unreachable 或者 ICMP Time Exceeded 错误报文，没有接收到相应错误报文的 IP 地址可被认为在该网络中。

2) 端口扫描

a) TCP 端口

- 基本扫描（使用 socket 直接进行 TCP 连接，connect 函数返回 0 时，连接成功）

  优点：实现简单、可用普通用户权限执行

  缺点：容易被目标应用日志记录

- 隐秘扫描
  - SYN 扫描（使用原始套接字，发送 SYN 包，若接收到 SYN+ACK 包，端口可连接）

    优点：一般不会被目标主机的应用所记录

    缺点：运行 Raw Socket 时必须拥有管理员权限
  - 发送 FIN 包

    • 对FIN报文的回复

    • TCP标准

    • 关闭的端口——返回RST报文

    • 打开的端口——忽略

    • BSD操作系统

    • 与TCP标准一致

    • 其他操作系统

    • 均返回RST报文

    优点：不会被记录到日志，可以绕过某些防火墙，netstat 命令不会显示（netstat 命令只能显示 TCP 连接或连接的尝试）

    缺点：使用 RAW IP 编程，实现起来相对比较复杂；不同操作系统结果不同，因此不完全可信
  - 其他

    • SYN + ACK 扫描

    • TCP null, Xmas

    • TCP Window

    • TCP ACK

    • FTP Proxy

    • idle

    • IP分段扫描

b) UDP 端口

攻击者向目标主机的 UDP 端口任意发送一些数据，如果这个 UDP 端口没有开放，则返回一个“目标不可达”ICMP 报文。

3) 系统类型扫描

6. 隐蔽扫描
   - 包特征随机化：TTL、源端口、目的端口。
   - 慢速扫描：很慢的速度来扫描对方主机。
   - 分片扫描：可以有效通过防火墙，并且不被日志。
   - 源地址欺骗：伪造源 IP 地址。
   - 使用跳板机：有效隐藏自己的 IP 地址。
   - 分布式扫描：一组攻击机共同对一台目标主机或者网络进行扫描。
7. 漏洞扫描
   1) 定义：利用一些专门或综合漏洞扫描程序对目标存在的系统漏洞或应用程序漏洞进行扫描
   2) 缺点：报告并不一定可靠，易暴露目标
   3) 策略：
      a) 被动式：基于主机的检测，对系统中不合适的设置、脆弱的口令以及其他同安全规则相抵触的对象进行检查
      b) 主动式：基于网络的检测，通过执行一些脚本文件对系统进行攻击，并记录它的反应，从而发现其中的漏洞
   4) 方法：直接测试、推断和带凭证的测试。
8. 网络拓扑探测
   1) 拓扑探测
      a) Traceroute 技术：tracert baidu.com，发现实际的路由路径
      b) SNMP
   2) 网络设备识别

第三章 口令攻击

1. 身份认证(Identification and Authentication)

   用户向计算机系统以一种安全的方式提交自己的身份证明，然后由系统确认用户的身份是否属实，最终拒绝用户或者赋予用户一定的权限。
2. 分类

   针对口令强度、口令传输、口令存储的攻击
3. 针对口令强度
   - 字典攻击、强力攻击、组合攻击、撞库攻击、彩虹表攻击
   - Windows 系统远程口令猜解：
   - 基于 IPC（共享“命名管道”的资源，它是为了让进程间通信而开放的命名管道）

     ```
     net use \\目标主机IP地址\ipc$ "口令" /user:"用户名"
     ```
4. 针对口令存储
   1) Windows 口令存储机制——SAM(Security account manager)数据库
      SAM 文件安全保护措施：

- 文件锁定：在操作系统运行期间，sam 文件被 system 账号锁定，即使用 admin 权限也无法访问它
- 隐藏：sam 在注册表中的备份是被隐藏的
- 不可读：系统保存 sam 文件时将 sam 信息经过压缩处理，因此不具有可读性

2) Windows 口令存储位置：

- %systemroot%system32\config\目录下
- %SystemRoot%\repair目录下的SAM._文件
- 注册表中HKEY_LOCAL_MACHINE\SAM键
- Winlogon.exe的内存块中

5. 针对口令传输

1) 嗅探：网卡处于混杂模式
2) 键盘记录：硬件/软件截获
3) 网络钓鱼

攻击者利用欺骗性的电子邮件和伪造的 Web 站点，骗取用户输入口令以及其他身份敏感信息。

4) 重放攻击

指攻击者记录下当前的通讯流量，以后在适当的时候重发给通讯的某一方，达到欺骗的目的。

6. 防范

1) 选择安全密码
足够长，混合使用各种字符
2) 防止口令猜测攻击
硬盘分区采用 NTFS 格式，正确设置和管理帐户
禁止不需要的服务，关闭不用的端口
3) 设置安全策略

第四章 缓冲区溢出

1. 漏洞攻击通常包含 3 个步骤：

漏洞发现（Vulnerability Discovery）、漏洞分析（Vulnerability Analysis）、漏洞利用（Vulnerability Exploit）。

2. 典型漏洞类型

- 栈溢出（Stack Overflow）
- 堆溢出（Heap Overflow）
- 格式化串（Format String）
- 整型溢出（Integer Overflow）
- 释放再使用（Use after Free）

3. 缓冲区：程序预留或分配一些逻辑上连续的数据空间用于对数据的缓存

4. 栈溢出

1) 基础

a) 当程序运行时，计算机会在内存区域中开辟一段连续的内存块，包括代码段、数据段和堆栈段三部分。

b) 堆栈段分为堆和栈。

- 堆（Heap）位于 BSS 内存段的上边，用来存储程序运行时分配的变量
  - 大小不固定，动态扩张或缩减
  - 内存释放由应用程序控制
- 栈（Stack）是一种用来存储函数调用时的临时信息的结构，如函数调用所传递的参数、函数的返回地址、函数的局部变量等。
  - 需要时分配，不需要时自动清除
  - 先进后出
  - 两个寄存器：SP/ESP（栈顶指针）、BP/EBP（基地址指针）

2) 程序调用时

a) 函数被调用时压入栈情况如下：

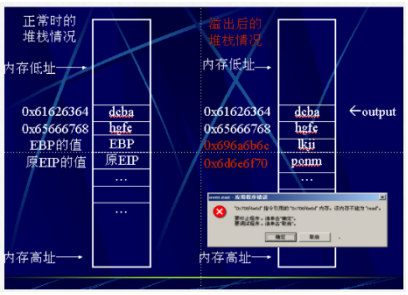
b) 在局部变量的下面，是前一个调用函数的 EBP，接下来就是返回地址。

c) 如果局部变量发生溢出，很有可能会覆盖掉 EBP 甚至 RET(返回地址)。

3) 原理：如果在堆栈中压入的数据超过预先给堆栈分配的容量时，就会出现堆栈溢出，从而使得程序运行失败，还有可能会导致系统崩溃。

函数调用时：

- EIP（下一条指令的地址）中的内容压入栈，作为函数的返回地址
- EBP（指向当前函数栈帧的底部）压入栈
- 当前栈指针 ESP 拷贝到 EBP 作为新的基地址，为本地变量预留空间，ESP 减去适当数值



5. 溢出攻击

1) 基本流程：

a) 注入恶意数据

b) 溢出缓冲区

c) 控制流重定向

d) 执行有效载荷

2) 利用过程：

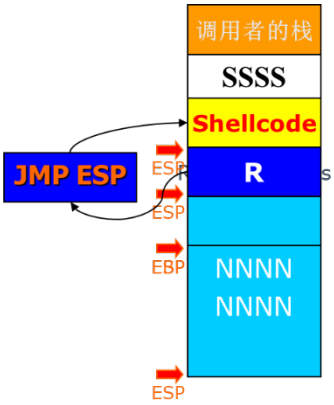
a) 有问题程序返回点的精确位置——我们可以把它覆盖成任意地址。
b) ShellCode——一个提供给我们想要的功能的代码。
c) JMP ESP 的地址——把返回点覆盖 JMP ESP 的地址

3) 关键技术：

a) 溢出点定位：
- 探测法：构造数据，根据出错的情况判断
- 反汇编分析

b) 覆盖执行控制地址
- 覆盖返回地址
- 覆盖函数变量指针
- 覆盖异常处理结构
  - 异常处理是一种对程序异常的处理机制，它把错误处理代码与正常情况下所执行的代码分开
  - 当程序发生异常时，系统中断当前线程，将控制权交给异常处理程序

c) 跳转地址的确定
- 跳转指令：jmp、esp、call ebx、call ecx 等
- 跳转指令搜索范围：用户空间的任意地址、系统 dll、进程代码段、PEB、TEB



d) shellcode 的定位和跳转
- shellcode 基本构成：
  - Nop Sled：类 NOP 指令填充，可以是 NOP，也可以是 inc eax 等无副作用指令。
  - Decoder：解码部分，对 Real_Shellcode 解码。
  - Real_Shellcode：真正有意义的 shellcode 部分，但是经过了编码处理。
- shellcode 定位
  - 方法一：NNNNNNSSSSSSSSRRRRRR
    适合大型缓冲区，“N”代表空指令，机器码为 0x90，“S”代表 shellcode，“R”代表覆盖的返回地址，即把返回地址 R 覆盖为 N 的大概位置，这样程序就会跳到 N 中执行，然后执行 shellcode。
  - 方法二：RRRRRRRRNNNNNNNNSSSS
    用大量的 R 填满整个缓冲区，然后大量的 Nop，最后 shellcode，这里 R 跳到 N 中，然后执行 shellcode。
  - 方法三：NNNNNRSSSSSS
    用系统核心 dll 中的 jmp esp 地址来覆盖返回地址，而把 shellcode 紧跟在后面，这样就可以跳转到我们的 shellcode 中了，R 为 jmp esp 地址

6. ShellCode 编写

1) shellcode：是一段能够完成一定功能(比如打开一个命令窗口)、可直接由计算机执行的机器代码，通常以十六进制的形式存在

2) 常见功能：

- 弹出对话框
- 打开 dos 窗口
- 添加系统管理用户
- 打开可以远程连接的端口
- 发起反向连接
- 上传（下载）木马病毒并运行
- 可能是攻击性的，删除重要文件、窃取数据
- 破坏，格式化磁盘

3) 字符与机器码转换表

Char	0	1	2	3	4	5	6	7	8	9		
Hex	30	31	32	33	34	35	36	37	38	39		
Char	A	B	C	D	E	F	G	H	I	J	K	L
Hex	41	42	43	44	45	46	47	48	49	4A	4B	4C
Char	M	N	O	P	Q	R	S	T	U	V	W	X
Hex	4D	4E	4F	50	51	52	53	54	55	56	57	58
Char	Y	Z	a	b	c	d	e	f	g	h	i	j
Hex	59	5A	61	62	63	64	65	66	67	68	69	6A
Char	k	l	m	n	o	p	q	r	s	t	u	v
Hex	6B	6C	6D	6E	6F	70	71	72	73	74	75	76
Char	w	x	y	z								
Hex	77	78	79	7A								

4) 常用功能编写

❖ Windows 函数调用原理：Func(argv1, argv2, argv3)

✧ 加载（LOAD）函数所在的动态链接库

✧ 将 argv3, argv2, argv1 压入堆栈

✧ 调用函数地址（Push EIP，JUMP FUNC）

✧ Push 是四个字节对齐的，因此必须每次压栈四个字节或者一个字节一个字节赋值

a) 打开对话框

➢ C 程序：

```
LoadLibrary("user32.dll");
MessageBox(NULL, "内容", "标题", MB_OK);
```

➢ 汇编+机器码

```
_asm{
    push ebp
    mov ebp,esp
    sub esp,0x40

    PUSH 标题机器码
    mov ebx,esp

    PUSH 内容机器码
    mov ecx,esp

    push 1
    push ebx            //标题
    push ecx            //内容
    xor eax,eax
    push eax
    push 0

    mov eax,77d507eah   //MessageBoxA()
    call eax
}
```

b) 打开 dos 窗口

- C 程序：LoadLibrary(“msvcrt.dll”);
  system(“command.com”);
- 汇编+机器码

```
push ebp ;
mov ebp,esp ;      把当前 esp 赋给 ebp
xor edi,edi ;
push edi ;压入 0，esp－4,;    作用是构造字符串的结尾\0 字符。
sub esp,08h ;加上上面，一共有 12 个字节,;用来放"command.com"。
mov byte ptr [ebp-0ch],63h ;  c
mov byte ptr [ebp-0bh],6fh ;  o
mov byte ptr [ebp-0ah],6dh ;  m
mov byte ptr [ebp-09h],6Dh ;  m
mov byte ptr [ebp-08h],61h ;  a
mov byte ptr [ebp-07h],6eh ;  n
mov byte ptr [ebp-06h],64h ;  d
mov byte ptr [ebp-05h],2Eh ;  .
mov byte ptr [ebp-04h],63h ;  c
mov byte ptr [ebp-03h],6fh ;  o
mov byte ptr [ebp-02h],6dh ;  m 一个一个生成串"command.com".
lea eax,[ebp-0ch] ;
push eax ;                    command.com 串地址作为参数入栈
mov eax, 0x7801AFC3 ;
call eax ;                  call system 函数的地址
```

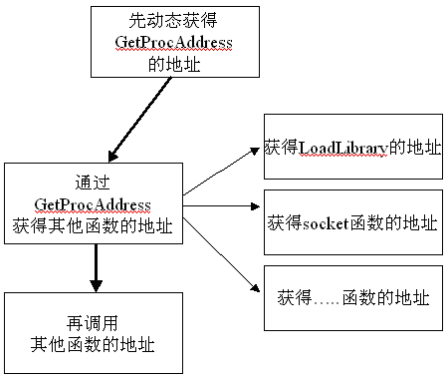
5) 通用 shellcode 编写

a) 正常退出：ExitProcess（）

```
xor eax, eax            //zero out eax (NULL)
push eax                // put zero to stack (exitcode parameter)
mov eax,0x7c81cafa      // ExitProcess(exitcode)
call eax                // exit cleanly
```

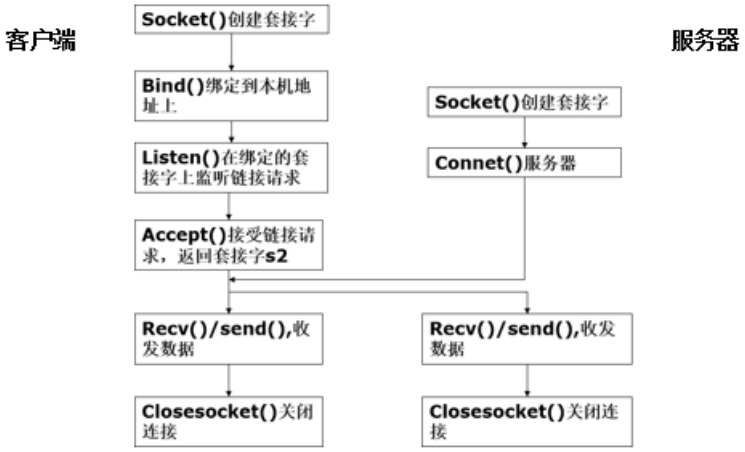
b) 动态定位函数地址：函数的返回值，通常都是放在 EAX 中

```
typedef void (*MYPROC)(LPTSTR);         //定义函数指针

int main()
{
HINSTANCE LibHandle;
MYPROC ProcAdd;
LibHandle = LoadLibrary("msvcrt.dll");
ProcAdd = (MYPROC) GetProcAddress(LibHandle, "system"); //查找 system 函数地址
(ProcAdd) ("command.com");          //其实就是执行 system("command.com")

return 0;
}
```



7. 后门 ShellCode 编写

1) 客户/服务器通信模型



2) 管道(PIPE)

a) 定义：一种简单的进程间通信(IPC)机制，一个进程往管道中写入数据，一个进程从管道中读取数据。

b) 分类：

- 命名管道：可以在同一台机器的不同进程之间以及不同机器之间的不同进程之间进行双向通信。
- 匿名管道：只能在父子进程之间或者一个进程的两个子进程之间进行单向通信。

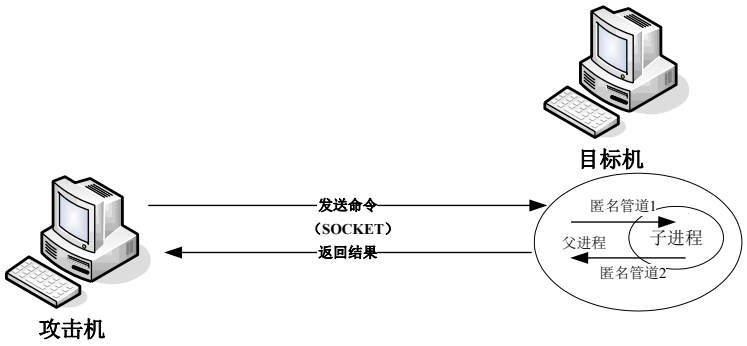
c) 匿名管道的使用

- BOOL CreatePipe(PHANDLE hReadFile, PHANDLE hWriteFile, LPSECURITY_ATTRIBUTES lpPipeAttributes, DWORD nSize)创建管道
- CreateProcess()创建子进程，并设置为继承
- WriteFile()往管道中写数据
- PeekNamedPipe()检查管道中是否有数据
- ReadFile()从管道中读取数据

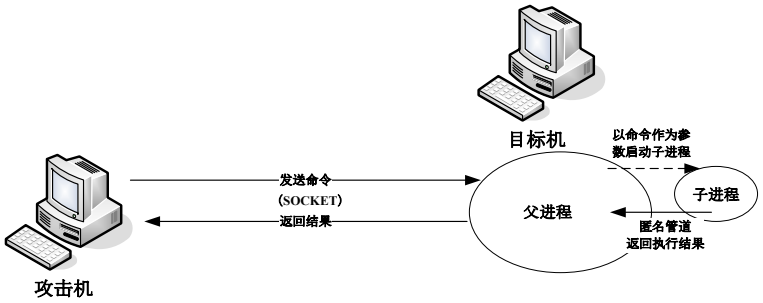
3) 后门

a) 模式：

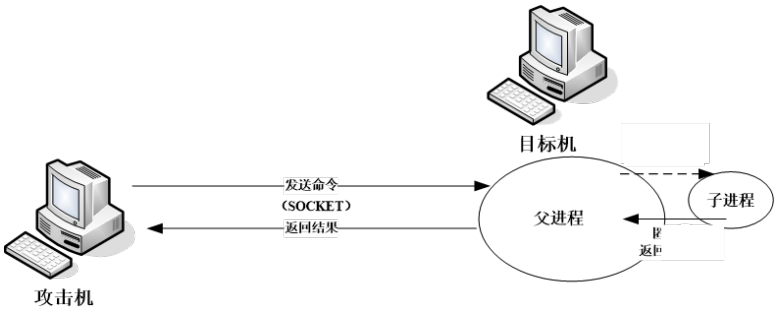
- 两个匿名管道



- 一个匿名管道



- 无管道：直接将子进程的输入和输出设置为 socket



b) 双管道、单管道、零管道、反连后门实现代码见 PPT

4) ShellCode 编码

a) 进行 ShellCode 编码的原因

- 避免 ShellCode 被截断
  避免 ShellCode 中出现”\x00”，从而被截断
- 符合目标程序的要求
  如 Foxmail 的 ShellCode 中不能有”/”，攻击 IIS 的 ShellCode 中不能有空格，即“\x20”
- 逃避 IDS、杀毒软件的检测

b) 检查 Exploit 是否失败

- 在 ShellCode 的最前面加上“\xEB\xFE”,即“JMP -1”的机器码，造成一个死循环。
- 按 Ctrl＋D 调出 SoftIce，会发现系统一直陷入这句指令不走。然后对照跟在后面的 ShellCode，看其是否和编写的一样，这样比较容易发现问题。

c) 异或法

- 编码--异或 97

  enShellCode[i]=ShellCode[i] ^ 0X97
- 解码—decode 程序
- ShellCode 拼接

```
AllShellCode[] =
//先是decode
"\xEB\x10\x5A\x4A\x33\xC9\x66\xB9\x00\x02“
"\x80\x34\x0A\x97\xE2\xFA\xEB\x05\xE8\xEB\xFF\xFF\xFF"
//后面跟enShellCode
"\xc2\x1c\x7b\xa4\x57\xc7\xc7\xc7\x51\xd2\x63\xda\x51\xd2\x62\xc4\x51\xd2\x61\xc1"
"\x51\xd2\x60\xd4\x51\xd2\x6f\xc5\x51\xd2\x6e\xc3\x51\xd2\x6d\xb9\x51\xd2\x6c\xd3"
"\x51\xd2\x6b\xdb\x51\xd2\x6a\xdb\x2d\xf3"
"\x8\x71\xe0\xc5\x1a\xd2\x63\xc7\x68\xc2"
"\x67\xc2\x1c\x7b\x14\x7b\xbb\x2f\xf4\xf8"
"\xfa\xfa\x1e\xd2\x63\x2f\xf6\xf9\xf3\xb9"
"\x1e\xd2\x6f\x2f\xf4\xf8\xfa\xb5\x1e\xd2"
"\x6b\xa4\x45\x1f\xc2\x68\x1a\xd2\x63\xc7"
"\x2f\x54\x38\x96\xef\x68\x47"
```

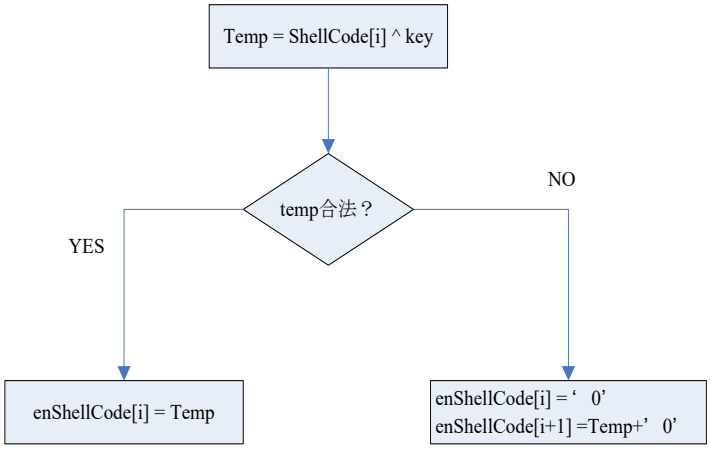
d) 微调法

- 原理: 在不改变指令功能的情况下个别改变代码(对不符合要求的字符进行等价指令变换)
- 如：IIS 漏洞中不能有 0X20，

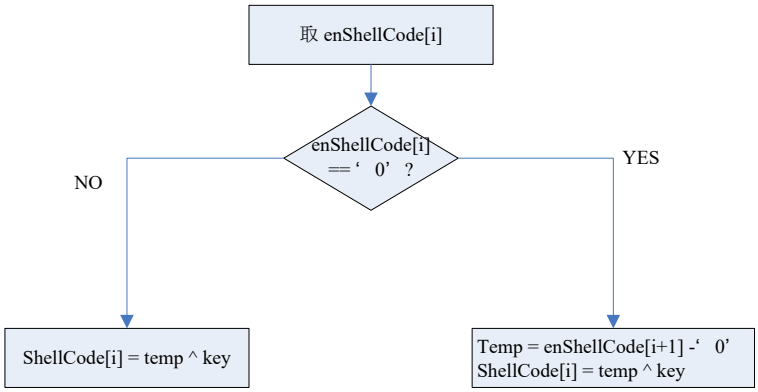
```
那么指令mov eax, 20h
改为: mov eax, 24h
      sub eax, 04h
```

e) 直接替换法

- 原理



- 解码



f) 字符拆分法

- Z=A+B
- 0xAB = 0xA * 0x10 + 0xB

g) 内存搜索法

- 主要针对：ShellCode 长度有限制的
- 原理：

AAAAAAAAAAAA Ret Search F ShellCode
AAAAAAAAAAAA' Ret Search F' ShellCode'

8. 堆溢出

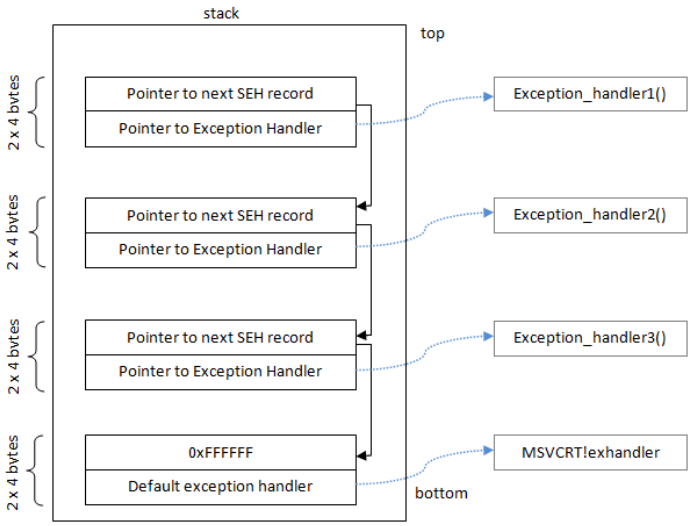
1) Windows 异常处理

a) 一个异常处理例程是内嵌在程序中的一段代码，用来处理在程序中抛出的异常，典型的异常处理例程如下所示。

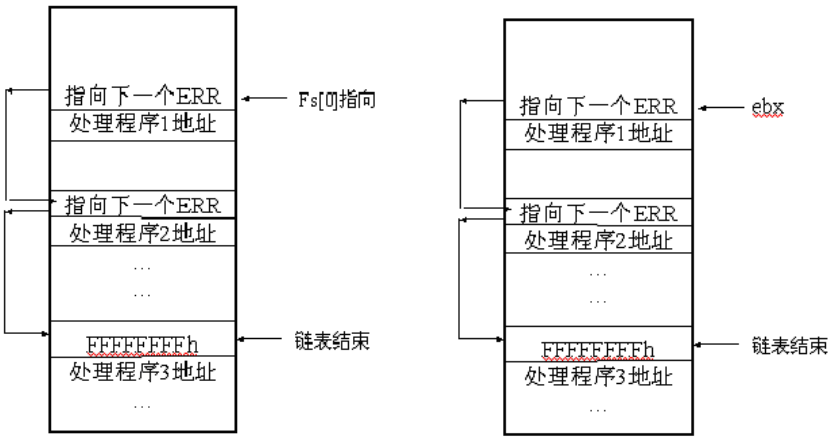
```
try
{
//run stuff. If an exception occurs, go to <catch>
code
}
catch
{
// run stuff when exception occurs
}
```

b) Windows 异常处理链表

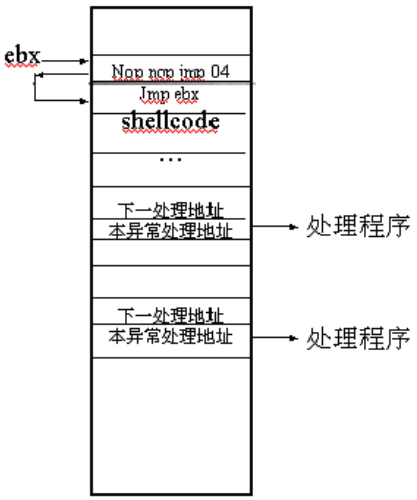
- SEH 记录
  - ✧ 基于帧的异常处理例程信息将以 exception_registration 结构储存在栈中
  - ✧ 大小为 8 个字节，有两个（4 byte）成员



- Windows 2000 中：判断是否使用现在指向的异常处理程序时，EBX 会自动变为下一处理点的地址，这样可以把异常处理地址连接起来，形成一个异常处理串



c) 覆盖异常



|----------||--------||----------|SSSSSSSSSS|
AAAAAAAA Jmp 04 JMP EBX地址 ShellCode

“AAA”的长度确定：

JMP ESP 改写成 JMP EBX，其思路是：先利用 JMP ESP 的攻击程序在 JMP ESP 代码后跟上“\xeb\xfe”，这句即 JMP –1。实行模拟攻击后，在被攻击机上调出 SoftICE，就会发现停在“\xeb\xfe”这句。查看其 fs:0000 的值，它里面存的就是异常处理的入口地址。计算那个值离现在的距离，就能知道要填充多少才能达到异常处理入口了。

d) ShellCode 定位

2) 堆溢出

a) 相关定义

- 堆(Heap)是 Windows 系统中的一种物理结构，用来动态分配和释放对象，用在事先不知道程序所需对象的数量和大小的情况下，或对象太大而不适合堆栈分配程序的时候。
- 堆栈，在可执行程序的 text 区，是从高地址向低地址扩展，是存放局部变量的地方，在编译时由编译器静态分配。
- 堆，是在可执行程序的 heap 区，从低地址向高地址扩展，是存放由 malloc 等函数动态分配数据的地方。还有其他的 data 区等。

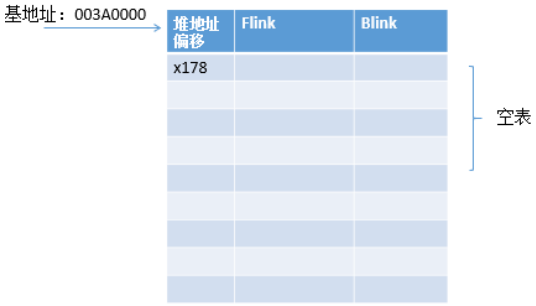
b) 堆的数据结构主要分为堆块和堆表两类

- 堆块
  - 堆块分为块首和块身。
  - 块首包含当前堆块的主要信息例如：此堆块的大小，是否是空闲态还是占用态等状态表信息。
  - 块身就是本堆块存放数据的位置，即最终分配给用户的数据区。块身位于块首的后面紧挨着。
- 堆表：空表和快表，空表：双向回环链表（空闲堆块信息表？），快表：单向链表

- ✧ 两种表都为 128 大小的指针数组 （空表每一项有两个指针，快表每一项有一个指针）
- ✧ 快表最多只有四个节点。快表的堆块处于占用状态，不会发生堆块合并。快表的只存在精确分配，快表优先空表分配。
- ✧ 空表除了数组的第一个元素外其他分别链接：数组下标*8 大小的堆块，数组的第一个元素链接着大于 1kb 的堆块，并升序排序

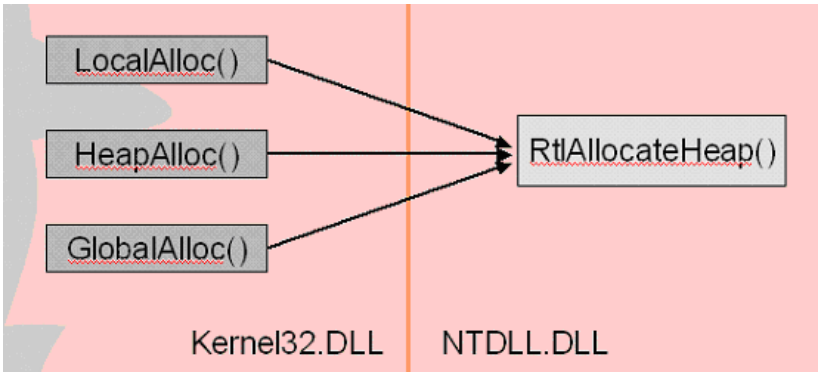
c) 堆初始化时的状态

- ➢ 其中只包含一个空闲大块（称为 “尾块”）
- ➢ 此尾块地址位于 0x178 处 （未启用块表的情况下）算上基地址就是 0x3A0178 (假设堆的基地址为 0x003A0000),又称为 freelist[0]。
- ➢ freelist[0] 指向“尾块”，八个字节 （前四个字节是前向指针 后四个字节是后向指针 即：空表中的一对指针），其余的各项索引都指向其自身。



d) 相关 API

- ■HeapCreate 创建一个新的堆对象
- ■HeapDestroy 销毁一个堆对象
- ■HeapAlloc 在堆中申请内存空间
- ■HeapFree 释放申请的内存
- ■HeapWalk 枚举堆对象的所有内存块
- ■GetProcessHeap 取得进程的默认堆对象
- ■GetProcessHeaps 取得进程所有的堆对象



3) 堆溢出

给分配的堆拷字符串时超过了所分配的大小，从而造成的溢出。

详细例子见 PPT、教材。

4) 堆溢出三部曲

a) 溢出点定位

b) ShellCode 编写

c) 跳转到 ShellCode

第五章 Web 应用攻击

1. Web 应用基础

1) 组成：

- Web 服务器
- Web 客户端
- HTTP 协议

2) Web 网页：

- 静态网页

  静态网页是指内容固定，不会根据 web 客户端请求的不同而改变的 web 网页
- 动态网页

  动态网页是相对于静态网页而言的，是指内容会根据时间、环境或用户输入的不同而改变的 Web 网页

3) 主流 Web 服务器

Apache、Microsoft IIS、Nginx

4) HTTP 协议

- HTTP1.0，支持 GET、POST 和 HEAD 方法
- HTTP1.1，该版本是当前最流行的 HTTP 协议版本

5) Web 应用攻击类型：

- web 客户端攻击（攻击用户）
  - 跨站脚本攻击（Cross-Site Scripting，简称 XSS 攻击）、
  - 网络钓鱼
  - 网页挂马
- Web 服务器攻击
  - 网页篡改
  - 代码注入攻击

- 文件操作控制攻击
- HTTP 头注入攻击
- HTTP 会话攻击

2. XSS 攻击

1) 定义：XSS 攻击是由于 Web 应用程序对用户输入过滤不足而产生的，使得攻击者输入的特定数据变成了 JavaScript 脚本或 HTML 代码

2) 同源策略：

A 网页设置的 Cookie，B 网页不能打开，除非这两个网页"同源"。所谓"同源"指的是"三个相同"：

- 协议相同
- 域名相同
- 端口相同

3) XSS 漏洞的危害

- 网络钓鱼，包括盗取各类用户账号
- 窃取用户 cookies 资料，从而获取用户隐私信息，或利用好用户身份进行一部对网站执行操作
- 劫持用户（浏览器）会话，从而执行任意操作，例如非法转账、强制发表日志、发送电子邮件等
- 强制弹出广告页面、刷流量等；
- 网页挂马；
- 进行恶意操作，例如任意篡改页面信息、删除文章等；
- 进行大量的客户端攻击，如 DDoS 攻击；
- 提取客户端信息，例如用户的浏览历史、真实 IP、开放端口等；
- 控制受害者机器向其它网站发起攻击；
- 结合其他漏洞，如 CSRF 漏洞，实施进一步作恶；
- 提升用户权限，包括进一步渗透网站；
- 传播跨站脚本蠕虫。

4) XSS 漏洞示例：

```
<head>
 <title>XSS示例程序</title>
</head>
<body>
 <h2>XSS示例程序</h2>
 <form action="t3.php" method="post">
  请输入您的名字:
  <input type="text" name="name" size="25">
  <input type="submit" value="递交">
 </form>
 </body>
```

```
<?php
if(!empty($_POST['name']))
{
  $name=$_POST['name'];
  setcookie("T2Cookie","1234567890",time()+3600*24);
  print("欢迎您，".$name);
}
?>
```

5) XSS 类型：

- 反射型 XSS

(1) 特点

a) 非持久性、参数型跨站脚本

b) 恶意脚本附加到 URL 地址参数中

(2) 工作流程：

(3) 防御方法：

隐藏网址内容

- 存储型 XSS

(1) 特点

a) 持久型

b) 一般攻击存在留言、评论、博客日志等中

c) 恶意脚本被存储在服务端的数据库中

(2) 工作流程

- DOM 型 XSS

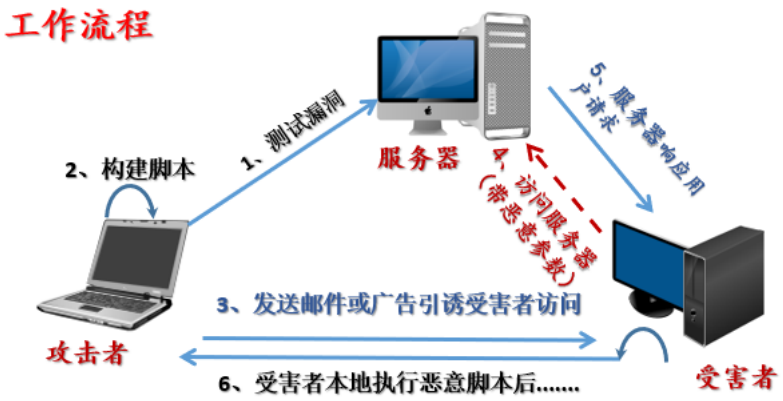
(1) 特点

a) DOM XSS 是基于在 js 上的

b) 不需要与服务端进行交互（在 URL 中即可操作）

c) 网站有一个 HTML 页面采用不安全的方式

(2) 工作流程



(3) 相关函数

- Document.write是把里面的内容写到页面里。
- document.URL是获取URL地址。
- Substring 从某处到某处，把之间的内容获取。
- document.URL.indexOf("name=")+5是在当前URL里从开头检索name=字符，然后加5(因为name=是五个字符，需要把他略去)，同时他也是substring的开始值
- document.URL.length是获取当前URL的长度，同时也是substring的结束值。
- 在URL获取name=后面的值，然后把name=后面的值给显示出来。

6) XSS 攻击

- Cookie 窃取

  Document.cookie
- 会话劫持
  - 会话 ID 由 Web 客户端提供给服务器以表示同一个会话，一般采用 Cookie 方式或 URL 方式传递。会话数据则一般保存在 Web 服务器，用于 Web 应用程序之间信息传递。
  - 会话劫持是指攻击者通过利用 XSS 攻击，冒用合法者的会话 ID 进行网络访问的一种攻击方式。
- 网络钓鱼
  - 攻击者可以执行 JavaScript 代码动态生成网页内容或直接注入 HTML 代码，从而产生网络钓鱼攻击。
  - 和传统的网络钓鱼攻击相比而言，通过 XSS 攻击实施网络钓鱼具有更强的隐蔽性。
- 信息刺探
  - 利用 XSS 攻击，可以在客户端执行一段 JavaScript 代码，因此攻击者可以通过这段代码实现多种信息的刺探。

- 访问历史信息、端口信息、剪贴板内容、客户端 IP 地址、键盘信息等。

- 网页挂马
  - 将 Web 网页技术和木马技术结合起来就是网页挂马。
  - 攻击者将恶意脚本隐藏在 Web 网页中，当用户浏览该网页时，这些隐藏的恶意脚本将在用户不知情的情况下执行，下载并启动木马程序。
- XSS 蠕虫
  - 一般利用存储型 XSS 攻击。
  - XSS 蠕虫的基本原理就是将一段 JavaScript 代码保存在服务器上，其他用户浏览相关信息时，会执行 JavaScript 代码，从而引发攻击

7) XSS 防范措施

- HttpOnly 属性

  指示浏览器禁止任何脚本访问 cookie 内容
- 安全编码

  小于号（<）转换成&lt、大于号（>）转换成&gt、与符号（&）转换成&amp、双引号（"）转换成&quot、单引号（'）转换成&#39

3. SQL 注入攻击

1) 原理

SQL 注入是由于 Web 应用程序对用户输入数据的合法性没有判断或过滤不严，攻击者可以在 Web 应用程序中事先定义好的查询语句的结尾上添加额外的 SQL 语句，以此来实现欺骗数据库服务器执行非授权的任意查询。

2) 常见 SQL 语法：

- 查询

  Select statement from table where condition
- 删除记录

  delete from table where condition
- 更新记录

  update table set field=value where condtion
- 添加记录

  insert into table field values(values)
- 常用函数

  Count()

  Asc(‘nchar’),unicode(‘nchar’)

  mid(str,n1,n2),substring(str,n1,n2)
- 联合查询

  select admin from where admin＝‘admin’ union select user from user

  注意 admin 字段的类型于 user 字段的类型必须匹配。
- 创建表、删除表

create table "table"("column1" "data type","column2" "datatype","column3" "data type");

drop table "table"

3) Sql 存储过程

- 系统存储过程

  以 sp_开头,用来进行系统的各项设定.取得信息.相关管理工作,如 sp_help 就是取得指定对象的相关信息

- 扩展存储过程：

  以 XP_开头,用来调用操作系统提供的功能

  exec master..xp_cmdshell 'ping 10.8.16.1

- 用户自定义的存储过程

4) SQL 注入攻击类型

- 基于错误信息 SQL 注入

  注入语句为：

  index.php?username=admin' and (extractvalue(1，concat(0x7e,(你想获取的数据的 sql 语句)))) and '1'='1

  index.php?username=admin' and (updatexml(1, concat(0x7e,(你想获取的数据的 sql 语句)),1)) and '1'='1

- SQL 盲注入

  为了防止基于错误信息的 SQL 注入，很多 Web 应用会将错误信息关闭，也就是通过网页看不到 Web 应用执行过程中的错误信息。SQL 盲注入就是在没有信息提示的情况实现 SQL 注入的方法。典型的 SQL 盲注入一般使用布尔值、时间函数等。

  注入步骤

  以 PHP+MYSQL（联合查询数值型注入）为例

  - 注入点的发现
  - 判断字段个数
  - 判断字段精确位置
  - 猜解数据库名用户名
  - 猜解表名字段名
  - 猜解内容

  登陆后台、上传木马、木马控制、提权

5) 一般方法示例

- 爆出数据库名

http://10.132.10.7/sql/sqli/example2.php?name=root' union select database(),2,3,4,5—sd

http://10.132.10.7/sql/sqli/example3.php?name=root'/**/union/**/select/**/database(),2,3,4,5/**/from/**/information_schema.columns%23

- 字段信息爆出

http://10.132.10.7/sql/sqli/example2.php?name=root' union select column_name,2,3,4,5 from information_schema.columns where table_name='users'—sd

http://10.132.10.7/sql/sqli/example3.php?name=root'/**/union/**/select/**/column_name,2,3,4,5/**/from/**/information_schema.columns/**/where/**/table_name='users'%23

- 账号密码爆出

http://10.132.10.7/sql/sqli/example2.php?name=root' union select name,passwd,3,4,5 from users—sd

http://10.132.10.7/sql/sqli/example2.php?name=root'/**/union/**/select/**/name,passwd,3,4,5/**/from/**/users%23

6) SQL 注入危害

- 读取、修改或者删除数据库内的数据，获取数据库中的用户名和密码等敏感信息
- 获得数据库管理员的权限
- 如果能够再利用 SQL Server 扩展存储过程和自定义扩展存储过程来执行一些系统命令，攻击者还可以获得该系统的控制权
- SQL 注入的隐蔽性：

  SQL 注入是从正常的 WWW 端口访问，防火墙一般不报警，很难发现

7) SQL 注入防范措施：

- 特殊字符转义
- 输入验证和过滤
- 参数化方法

4. 文件上传

1) 文件上传绕过产生的原因

- 文件名未作安全处理。
- 文件内容、类型未过滤。
- 文件特性特征未处理。

2) 常见的绕过方式

- javascript 验证突破
- php 文件后缀解析绕过
- Windows 解析缺陷
- 00 截断绕过
- 特别文件名构造

5. 文件包含

1) 介绍：

文件包含漏洞本质是服务器利用文件包含特性，利用相应的函数功能调用或者获取

文件，从而执行命令。这些都是网站编写者对于功能过滤不严谨导致的结果。

2) 产生原因：

PHP 文件包含漏洞的产生原因是在通过 PHP 的函数引入文件时，由于传入的文件名没有经过合理的校验，从而操作了预想之外的文件，就可能导致意外的文件泄露甚至恶意的代码注入。

3) 常见涉及函数

- include()
- include_once()
- require()
- require_once()
- 示例：

(1) <?php

```
$file = $_GET['file'];
include($file);
```

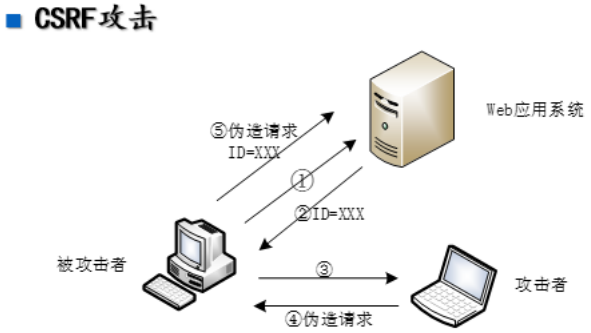
?>

(2) <?php phpinfo();?>

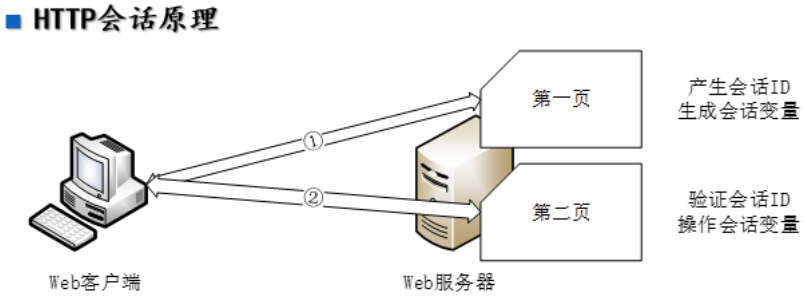
6. HTTP 会话攻击及防御

1) HTTP 会话攻击技术有

- 预测会话 ID
- 窃取会话 ID
- 控制会话 ID
- 跨站请求伪造攻击（Cross-Site Request Forgery，CSRF）



2) 原理



3) 防范措施

- 针对预测会话 ID 号攻击

  采用编程语言内置的会话管理机制，如 PHP 语言、JAVA 语言的会话管理机制等

- 针对窃取会话 ID 号攻击

  需要根据不同的窃取会话 ID 号方法，采取不同的防范措施，如基于 XSS 攻击实施的会话 ID 号窃取攻击，可以采用 HttpOnly 属性的方法来防范

- 针对会话 ID 固定攻击

  支持会话采纳（Session Adoption）的 Web 环境，存在会话 ID 号固定的风险比较高。因此，尽可能的采用非会话采纳的 Web 环境或对会话采纳方式进行防范。

- 针对会话保持攻击

  不能让会话 ID 号长期有效，如采用强制销毁措施或用户登录后更改会话 ID 号等

- 针对 CSRF 攻击

  (1) 使用 POST 替代 GET

  (2) 检验 HTTP referer

  (3) 验证码

  (4) 使用 Token

7. 远程代码执行（RCE）

1) 原理

远程代码执行是指用户通过浏览器提交执行命令，由于服务器端没有针对执行函数做过滤，导致在没有指定绝对路径的情况下就执行命令，可能会允许攻击者通过改变 $PATH 或程序执行环境的其他方面来执行一个恶意构造的代码。

2) 方法：在一些特定的函数中，通过构造特定 php 代码语句，执行恶意语句，达到目的的作用。

常见的 php 代码注入的函数有：

eval()，assert()，system()，preg_replace()，create_function，call_user_func，call_user_func_array，array_map()，反引号，exec()。

如果程序中使用了以上这些函数，而且提交的变量可控就有可能导致代码注入漏洞。

8. 序列化与反序列化

1) 原理

在很多应用中，需要对某些对象进行序列化，让它们离开内存空间，入住物理硬盘，以便长期保存。比如最常见的是 Web 服务器中的 Session 对象，当有 10 万用户并发访问，就有可能出现 10 万个 Session 对象，内存可能吃不消，于是 Web 容器就会把一些 seesion 先序列化到硬盘中，等要用了，再把保存在硬盘中的对象还原到内存中。

2) 序列化的两种用途：

- 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中或数据库中，如 Session。
- 在网络上传送对象的字节序列，可以省去将对象实例化的繁琐操作。

3) 举例：

■以php序列化和反序列化举例说明。创建一个$arr数组用于储存用户基本信息，通过var_dump函数查看其结构。

```
$arr=array();
$arr['name']='张三';
$arr['age']='22';
$arr['sex']='男';
$arr['phone']='123456789';
$arr['address']='上海市浦东新区';
var_dump($arr);
```

■打印输出：

```
array(5) {
    ["name"]=> string(6) "张三"
    ["age"]=> string(2) "22"
    ["sex"]=> string(3) "男"
    ["phone"]=> string(9) "123456789"
    ["address"]=> string(21) "上海市浦东新区"
        }
```

9. 暴力破解

1) 常见形式

- 固定账号对密码暴力破解。
- 在得知账号具有规律性，或者通过某种方式获取到大量账号的前提下，固定密码对账号暴力破解。
- 使用网上流传的账号密码库进行撞库攻击。

10. 逻辑漏洞

1) 登录时常见逻辑漏洞

- 返回包中有验证码。
- 返回页面 hidden 中有验证码。
- 有些其他登陆 url 中不需要验证码。
- 验证码不变，验证码没有一个完整的服务请求，只在刷新 url 时才变。
- 第一次请求包验证了验证码是否正确，第二次请求不需要验证。
- 拦截登录时验证码的刷新请求，第一次验证码未失效，可绕过
- 验证码和用户名、密码是否一次同时提交
- 公众号，app 无验证

(一) 身份认证

i. 破解方法：

- 使用已知用户名对密码进行暴力破解
- 使用一个弱口令密码对用户进行暴力破解

ii. 破解方法：

- Burpsuite
- Hydra

2) Cookie 和 Session 问题

Cookie 机制采用的是在客户端保持状态的方案，用来记录用户的一些信息，也是实现 Session 的一种方式。

Session 机制采用的是在服务器端保持状态的方案，用来跟踪用户的状态，可以保存在集群、数据库、文件中。

- 一些网站会利用 Cookie 是否为空、Session 是否为 true 来判断用户是否可以登录，只要构造一个 Cookie 或 Session 为 true 就可以绕过认证登录。
- Session 会话固定攻击，一种诱骗受害者使用攻击者指定的会话标识（Session id）的攻击手段。攻击者通过某种手段重置目标用户的 Session id，然后监听用户会话状态。
- Cookie 仿冒攻击，通过修改 Cookie 中的某个参数来实现登录其他用户。

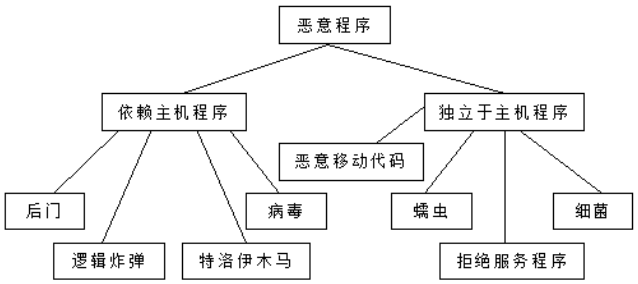
(二) 数据篡改

1. 数据篡改形式

- 手机号篡改
- 邮箱或者用户篡改
- 订单 ID 篡改
- 商品编号篡改

第六章 恶意代码

1. 定义：未经授权、干扰或破坏计算机系统/网络功能的程序或代码（一组指令：二进制文件/脚本语言）。
2. 功能：
   1) 删除敏感信息
   2) 网络传播的起点
   3) 监视键盘
   4) 收集主机用户相关信息
   5) 获取屏幕
   6) 在系统上执行指令/程序
   7) 窃取文件
   8) 开启后门
   9) 隐藏其在主机上的所有活动
3. 类型
   1) 计算机病毒
   2) 特洛伊木马
   3) 计算机蠕虫
   4) 逻辑炸弹
   5) 用户级 RootKit
   6) 内核级 RootKit
   7) 脚本恶意代码
   8) 恶意 ActiveX 控件
   9) 其它恶意代码



4. 计算机病毒
   1) 生命周期：潜伏→传染→触发→发作
   2) 特征
      a) 传染性
      b) 非授权性
      c) 隐蔽性
      d) 潜伏性
      e) 破坏性
      f) 不可预见性
      g) 可触发性
   3) 三种机制

a) 传染机制

- 寄生感染：病毒将其代码放入宿主程序中
- 插入感染和逆插入感染
- 破坏性感染
- 滋生/伴侣感染

b) 触发机制

- 日期和时间触发
- 键盘触发
- 感染触发
- 启动触发
- 磁盘访问触发和中断访问触发
- 其他触发：OS 型号、IP 地址、语言、地区、特定漏洞

c) 破坏机制

- 攻击系统数据区
- 攻击文件和硬盘
- 攻击内存
- 干扰系统运行
- 扰乱输出设备
- 扰乱键盘

5. 蠕虫

1) 定义：计算机蠕虫可以独立运行，并能把自身的一个包含所有功能的版本通过网络传播到另外的计算机上。（副本的完整性和独立性）

2) 使用蠕虫使得追踪困难，并能发动 DDOS 攻击

3) 行为特征：

a) 主动攻击
b) 行踪隐蔽
c) 利用系统、网络应用服务漏洞
d) 造成网络拥塞
e) 降低系统性能
f) 产生安全隐患
g) 反复性/破坏性

4) 组成

a) 弹头：如何获得控制权
b) 传播引擎
c) 目标算法
d) 扫描引擎
e) 有效载荷

5) 工作流程：随机生成 IP 地址→探测地址→主机是否存在→漏洞是否存在→攻击、传染和现场处理

6) 病毒和蠕虫的区别

	病毒	蠕虫
存在形式	寄生	独立个体
复制机制	插入到宿主程序（文件）中	自身的拷贝
传染机制	宿主程序运行	系统存在漏洞（vulnerability）
搜索机制（传染目标）	针对本地文件	针对网络上的其它计算机
触发传染	计算机使用者	程序自身
影响重点	文件系统	网络性能、系统性能
计算机使用者角色	病毒传播中的关键环节	无关
防治措施	从宿主文件中摘除	为系统打补丁（Patch）
对抗主体	计算机使用者、反病毒厂商	系统提供商、网络管理人员

6. 后门：允许攻击者绕过系统中常规安全控制机制的程序

7. 特洛伊木马

1) 定义：特洛伊木马是一种基于远程控制的黑客工具（实质是 C/S 结构的网络程序）

2) 特点：欺骗性、隐蔽性、非授权性

3) 功能

a) 保留访问权限
b) 信息收集
c) 远程控制
d) 远程命令执行
e) 远程文件操作
f) 其它的特殊功能

4) 木马 VS 病毒/蠕虫

a) 相同：传播渠道、窃取信息功能
b) 不同：隐蔽性、不主动传播、远程控制

5) 木马 VS 远程管理软件

a) 相同：C/S 结构、强大的远程控制功能
b) 不同：隐蔽性、功能特殊性

6) 植入技术

a) 直接攻击法
b) 文件下载法
c) 解除植入法
d) 电子邮件传播法
e) 网页挂马法
f) 应用软件漏洞法
g) 社会工程学法

7) 木马程序开发实例（自己看）

8) 检测与防范

a) 木马启动

- 修改批处理
- 修改系统配置
- 借助自动运行功能
- 通过注册表中的 Run 来启动
- 通过文件关联启动
- 通过 API HOOK 启动
- 通过 VXD 启动
- 通过浏览器网页启动
- 利用 Java applet
- 利用系统自动运行的程序

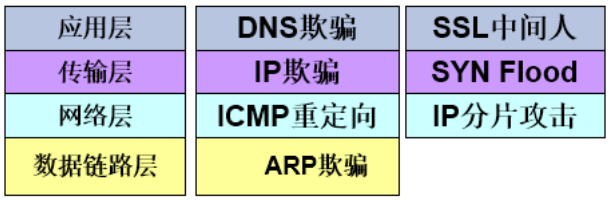
b) 检测及清除

- 查看注册表中的可疑键值，将其删除
- 利用开放主机端口号和各个木马程序使用端口的对应关系，判断主机是否已中木马，中了何种木马，并能根据所中木马的类型，对其进行杀灭

第七章 假消息攻击

1. 定义：利用网络协议设计中的安全缺陷，通过发送伪造的数据包达到欺骗目标、从中获利的目的

2. 类型：



3. 网络嗅探

截获网络中传输的数据，并从中解析出其中的机密信息。

4. ARP 欺骗攻击

1) 原理：

ARP 请求是一种广播包，为了避免在网络中出现过多的广播，提高网络的效率，每台主机使用一个数据结构保存最近使用过的 MAC 地址，即 ARP 缓存。

攻击者只要持续不断的发出伪造的 ARP 响应包就能更改目标主机 ARP 缓存中的 IP-MAC 条目，造成网络中断或中间人攻击。

2) 危害：拒绝服务攻击、嗅探、中间人攻击

3) 防范：

a) 建立 DHCP 服务器
b) 局域网中建立 MAC 数据库
c) 网关关闭 ARP 动态刷新的过程，使用静态路由
d) 局域网监听 ARP 数据包
e) 使用 VLAN 或 PVLAN 技术

5. ICMP 重定向攻击

1) 原理：

ICMP 重定向信息就是路由器向主机提供实时的路由信息，当一个主机收到 ICMP 重定向信息时，它就会根据这个信息来更新自己的路由表。

但由于缺乏必要的合法性检查，如果攻击者想要被攻击机修改它的路由表，发送 ICMP 重定向信息给被攻击的主机，就可以让该主机按照要求来修改路由表。

2) 条件：

a) 新路由必须是直达的
b) 重定向包必须来自去往目标的当前路由
c) 重定向包不能通知主机用自己做路由
d) 被改变的路由必须是一条间接路由

3) 危害：

a) 改变对方的路由表
b) 拒绝服务攻击、嗅探、中间人攻击

4) 防范

a) 配置防火墙，拒绝接收 ICMP 重定向报文
b) 在 Linux 下可以通过在防火墙上拒绝 ICMP 重定向报文或者是修改内核选项重新编译内核来拒绝接收 ICMP 重定向报文
c) 在 Windows 下可以通过防火墙和 IP 策略拒绝接收 ICMP 报文

6. IP 欺骗攻击

1) 危害：

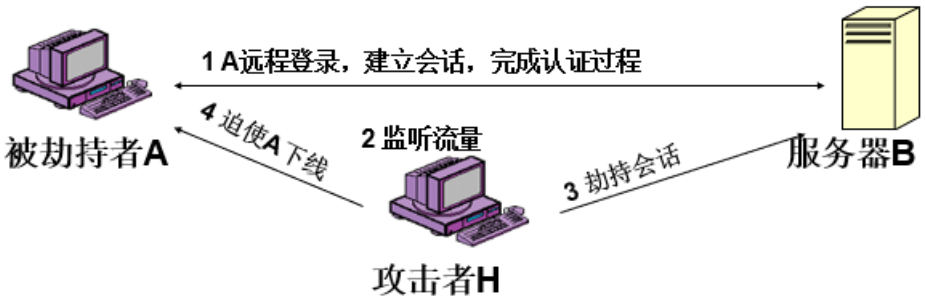
a) 以可信任的身份与服务器建立连接
b) 伪造源 IP 地址，隐藏攻击者身份，消除攻击痕迹

2) 防范：

a) 在终端会话变成不活动状态之前立刻注销登录，并且仅在需要时启动终端会话
b) 使用基于加密的终端协议
c) 使用随机化的初始序列号，使得 TCP 序列号难以猜测

d) 抛弃基于地址的信任策略

7. 会话劫持



TCP 会话劫持

1) 关键：预测正确的序列号
2) 步骤
   a) 发现目标
   b) 探查远程机器的 ISN（初始序列号）规律
   c) 等待或者监听会话（最好在流量高峰期进行，不容易被发现，而且可以有比较多可供选择的会话）
   d) 猜测序列号
   e) 使被劫持方下线（通过 ACK 风暴或拒绝服务）
   f) 接管会话
3) TCP ACK 风暴
   a) 当一个主机接收到一个不期望的数据包的时候，它会用自己的序列号发送 ACK，而这个包本身也是不可被接受的。于是，两边不停地发送 ACK 包，形成 ACK 包的循环，是为 ACK 风暴。
   b) 如果有一个 ACK 包丢掉，则风暴停止。

8. DNS 欺骗攻击

1) 原理
   a) 客户端以特定的标识 ID 向 DNS 服务器发送域名查询数据包
   b) DNS 服务器查询之后以同样的 ID 返回给客户端响应数据包
   c) 攻击者拦截该响应数据包，并修改其内容，返回给客户端
2) 危害
   a) 将用户访问的合法网址重定向到另一个网址
   b) 使用户在不知情的情况下访问恶意网站
3) 防范
   a) 构建静态的域名-IP 映射，如 Windows 系统的 hosts 文件
   b) 直接用 IP 地址连接服务器

9. SSL 中间人攻击

1) 防范：

a) 仔细查看证书的来源和属性

b) 不要打开证书来源不可信的网址

第八章 拒绝服务攻击

1. 拒绝服务攻击概念

攻击者通过攻击网络节点、网络链路或网络应用，致使合法用户无法得到正常服务响应的网络攻击行为。

2. 分类

1) 漏洞型拒绝服务攻击：指利用软件实现中存在的漏洞，致使服务崩溃或者系统异常。

如：Ping of Death; Tear of Drop

2) 重定向拒绝服务攻击：利用网络协议的设计缺陷，使目标传输的数据被重定向至错误的网络地址，从而无法进行正常的网络通信。

如：ARP 欺骗、DNS 欺骗

3) 资源消耗型拒绝服务攻击：通过大量的请求占用网络带宽或系统资源，从而导致服务可用性下降甚至丧失。

如：SYN FLOOD、UDP FLOOD、HTTP FLOOD

3. 典型拒绝服务攻击

1) 传统拒绝服务攻击

- Ping of Death：针对存在漏洞的 Windows 95、WinNT、Linux 2.0.x 等系统，通过向其发送超长的 IP 数据包导致目标系统拒绝服务。
- Tear of Drop：利用 IP 包的分片重组在多个操作系统协议栈中实现时存在的漏洞。
- Land 攻击：发送一个伪造的 TCP SYN 报文，源地址和目的地址设置均为目标 IP 地址，源端口和目标端口设置为目标某个开放的 TCP 端口，可以导致目标主机死锁。

2) 洪泛攻击

- 共同特征：发送大量数据包，迫使目标服务消耗大量资源来处理无用请求。
- 根据攻击发生的协议层次，可分为网络层洪泛、传输层洪泛和应用层洪泛等。

(1) TCP 洪泛（SYN FLOOD）：通过发送大量伪造的 TCP 连接请求，致使目标服务 TCP 连接资源被耗尽的拒绝服务攻击。

(2) UDP 洪泛（UDP Flood）：通过发送大量的 UDP 报文，消耗目标服务器带宽资源的一种拒绝服务攻击。

Echo/Chargen 攻击，反射型分布式拒绝服务攻击多为 UDP 洪泛攻击。

(3) HTTP 洪泛（HTTP Flood）：利用大量看似合法的 HTTP GET 或 POST 请求消耗 Web 服务器资源，最终导致其无法响应真正合法的请求。

3) 低速率拒绝服务（LDoS, Low-rate Denial-of-Service）攻击

- 利用网络协议或应用服务协议中的自适应机制存在的安全问题，通过周期性地发送高速脉冲攻击数据包，达到降低被攻击主机服务性能的目的。

  TCP 拥塞控制-RTO

  发送端为发送的每个报文设置一个定时器，若收到报文的确认之前定时器超时将重新发送该报文，这里设置的定时器就是 RTO。

  TCP 拥塞控制-AIMD

  发送端在收到 3 个重复的 ACK 数据包时就开启重传，启动 AIMD 算法调整拥塞窗口大小。

- 原理：LDoS 攻击利用 TCP 拥塞控制机制，故意制造网络拥塞状况，使拥塞控制一直处于调整状态，发送端的发送速率会迅速变小，导致被攻击主机的服务性能显著降低。

4. 分布式拒绝服务攻击

1) 采用协作的方式，利用网络中位置分布的大量主机向目标发起的拒绝服务攻击。按协同方式不同分为：

2) 基于僵尸网络的 DDoS

- 僵尸网络：是攻击者出于恶意目的，传播僵尸程序控制大量主机，并通过一对多的命令与控制信道所组成的网络。
- 集中式命令控制机制

  僵尸节点通过连接到一个或多个控制服务器来获取控制信息或命令。基于 IRC 协议的僵尸网络和基于 HTTP 协议的僵尸网络都属于集中式命令控制机制的僵尸网络。
- 分布式命令控制机制

  在使用分布式命令控制机制时，攻击者通常会任意地连接到某个僵尸节点，在该节点上发布命令控制信息，命令会采用 Push 或 Pull 的方式在整个僵尸网络中传递。
- 攻击者借助僵尸网络发动 DDoS 攻击通常需要经过以下几个阶段：

  (1) 构建僵尸网络

  (2) 收集目标信息

  (3) 实施 DDoS 攻击

3) 反射型分布式 DDoS

- RDDoS 攻击中，攻击者利用反射器将大量的响应包汇集到受害者主机，导致拒绝服务。

(1) DNS 反射攻击通过向 DNS 服务器发送伪造源地址的查询请求将应答流量

导向攻击目标，亦称为 DNS 放大攻击。

(2) LDAP 放大攻击通过向 LDAP（Lightweight Directory Access Protocol，轻量目录访问协议）服务器发送伪造源地址的查询请求来将应答流量导向攻击目标。

(3) NTP 放大攻击通过向 NTP（Network Time Protocol，网络时间协议）服务器发送伪造源地址的查询请求将应答流量导向攻击目标。

5. 拒绝服务攻击防御

1) 从部署位置的角度可以分为源端防御、目标端防御、中间网络防御和混合防御

2) 从技术的角度，拒绝服务攻击的防御可以分为预防、检测、响应与容忍

- 预防：
  (1) 抑制僵尸网络规模
  (2) 过滤伪造源地址报文
  (3) 减少可用反射/放大器
- 检测：
  (1) 特征检测：通过分析得到攻击行为区别于其它正常用户访问行为的唯一特征，并据此建立已知攻击的特征库。
  (2) 异常检测：攻击会导致当前的网络状态与正常网络状态模型产生显著的不同。异常检测通过对比两者检测出攻击的发生。
- 响应：
  (1) “流量清洗”：对攻击流量进行过滤，设法将恶意的网络流量剔除掉，只将合法的网络流量交付服务器。
- 容忍：通过提高处理请求的能力来消除攻击的影响
  (1) CDN（内容分发网络）：在互联网范围内广泛设置多个节点作为代理缓存，并将用户的访问请求导向最近的缓存节点，以加快访问速度。
  (2) AnyCast（任播）：一种网络寻址和路由方法。一组提供特定服务的服务器可以使用相同的 IP 地址，提供相同的服务。