第7章 网络与系统攻击技术

中国科学技术大学
曾凡平
billzeng@ustc.edu.cn

课程回顾：第6章 主机系统安全技术

6.1 操作系统安全技术	• 6.1.1 基本概念 • 6.1.2 可信计算机评价标准(TCSEC) • 6.1.3 操作系统安全的基本原理 • 6.1.4 操作系统安全机制 • 6.1.5 Linux的安全
6.2 数据库安全技术	• 6.2.1 传统数据库安全技术 • 6.2.2 外包数据库安全 • 6.2.3 云数据库/云存储安全
6.3 可信计算技术	• 6.3.1 概念和基本思想 • 6.3.2 TCG可信计算系统

第7章 网络与系统攻击技术

7.1 网络攻击概述

- 网络攻击的概念，网络攻击的一般流程

7.2 网络探测

- 网络踩点，网络扫描，常见的扫描工具

7.3 缓冲区溢出攻击

- 缓冲区溢出的基本原理，缓冲区溢出的防范

7.4 拒绝服务攻击

- 常见的拒绝服务攻击
- 分布式拒绝服务攻击，拒绝服务攻击的防范

7.5 僵尸网络

7.6 缓冲区溢出漏洞的分析与利用

第7章 网络与系统攻击技术

- 网络与系统攻击技术，即针对企业网络或重要计算机系统进行攻击的技术。
- 网络与系统攻击是利用网络与系统中存在的**漏洞和缺陷**实施入侵和破坏，实质上是一系列方法和手段的集合。
- 近几年来，随着计算机网络的迅速普及和网络技术的快速发展，攻击手段也在不断发展变化。

7.1 网络攻击概述

- 目前造成网络不安全的主要因素是系统、协议及数据库等在设计上存在漏洞和缺陷。漏洞的3种常见定义：

① 在计算机安全中，（漏洞）是系统安全步骤、管理控制、内部控制等处的一个缺陷，有可能被攻击者利用获取对信息的非授权访问或破坏关键任务；

② 在计算机安全中，（漏洞）是物理设施、人员、管理、硬件或软件上的缺陷，这些缺陷可能被利用并给系统带来危害。漏洞是一种或一组可能使系统受到攻击的状态。

③ 在计算机安全中，（漏洞）是系统中存在的任何错误或缺陷。

- 一般认为，**软件漏洞是**软件设计或配置中**能被攻击者利用的错误或缺陷**。

7.1.1 网络攻击的概念

- **网络攻击是指攻击者利用网络存在的漏洞和安全缺陷对网络系统的硬件、软件及其系统中的数据进行的攻击。**

① 目前网络互连一般采用TCP/IP，它是一个工业标准的协议簇，但该协议簇在制定之初对安全问题考虑不多，协议中有很多的安全漏洞。

② 同样，数据库管理系统(DBMS)也存在数据的安全性、权限管理及远程访问等方面问题。在DBMS或应用程序中可以预先安置从事情报收集、受控激发、定时发作等破坏程序。

③ 如果应用软件设计不当，恶意用户可以通过输入特定的数据达到攻击目的。

网络攻击的概念

- 由此可见，针对系统、网络协议及数据库等，无论是其自身的设计缺陷，还是由于人为的因素产生的各种安全漏洞，都可能被一些另有图谋的攻击者所利用并发起攻击。

- 因此，若要保证网络安全可靠，必须熟知攻击者实施网络攻击的技术原理和一般过程。只有这样，才能做好必要的安全防备，从而确保网络运行的安全和可靠。

7.1.2 网络攻击的一般流程

- 攻击者在实施网络攻击时的一般流程包括系统调查、系统安全缺陷探测、实施攻击、巩固攻击成果和痕迹清理五个阶段。

(1)系统调查：

- 攻击者选取攻击目标主机后，利用公开的协议或工具**通过网络收集目标主机相关信息的过程**。例如，攻击者可以通对SNMP查阅网络系统路由器的路由表从而了解目标主机所在网络的拓扑结构及其内部细节。

- 这个阶段实质上是一个信息搜集的过程，这一过程并不对目标主机产生直接的影响，而是为进一步的入侵提供有用信息。

网络攻击的一般流程

(2)系统安全缺陷探测：

- 在收集到攻击目标的相关信息后，攻击者通常会利用一些自行编制或特定的软件扫描攻击目标，**寻找攻击目标系统内部的安全漏洞**，为实施真正的攻击做准备。
- 扫描采取模拟攻击的形式对目标可能存在的已知安全漏洞逐项进行检查，目标可能是工作站、服务器、路由器、交换机等，根据扫描结果向攻击者提供周密可靠的分析报告。

- **步骤(1)和(2)也称为网络探测。**

网络攻击的一般流程

(3)实施攻击：

- 当获取到足够的信息后，攻击者就可以结合自身的水平及经验总结制定出相应的攻击方法，**实施真正的网络攻击**。这一阶段的核心任务是解决如何进入目标系统的问题，常用手段包括破解口令直接获取权限，或利用软件漏洞植入恶意软件。

(4)巩固攻击成果：

- 基于前期的侵入结果，控制目标系统，完成既定攻击任务。这一阶段的**重点是长期隐蔽潜伏**，并完成攻击任务。常见手段是利用木马等控制软件。

(5)痕迹清理：

- 在成功实施攻击后，攻击者往往会利用获取到的目标主机的控制权，清除系统中的日志记录和留下后门，**消除攻击过程的痕迹**，以便日后能不被觉察地再次进入系统。

7.2 网络探测

- **网络探测也称为网络侦察**。

- 由于初始信息的未知性，网络攻击通常具备一定的难度。因此，探测是攻击者在攻击开始前必需的情报收集工作，攻击者通过这个过程需要**尽可能多地了解攻击目标安全相关的各方面信息**，以便能够实施针对攻击。

- 探测可以分为三个基本步骤：踩点、扫描和查点。

7.2.1 网络踩点

网络踩点(footprinting)

- 是指攻击者收集攻击目标相关信息的方法和步骤，主要包括攻击对象的各种联系信息，包括名字、邮件地址、电话号码、传真号、IP地址范围、DNS服务器、邮件服务器等相关信息。其目的在于了解攻击目标的基本情况、发现存在的安全漏洞、寻找管理中的薄弱环节和确定攻击的最佳时机等，为选取有效的攻击手段和制定最佳的攻击方案提供依据。
- 对于一般用户来说，如果能够利用互联网中提供的大量信息来源，就能逐渐缩小范围，从而锁定所需了解的目标。目前实用的流行方式有：通过网页搜寻和链接搜索、利用互联网域名注册机构进Whois查询、利用，Traceroute获取网络拓扑结构信息等。

踩点方法一:
传统方法，用whois通过因特网实施踩点

- 最常规的工具是whois查询工具，它是Linux系统内置的查询工具，可以把企业的很多在线信息查出来，其中包括：
  - ✧Internet Registrar数据(企业申请上网时填报的信息)
  - ✧企业各职能部门的组织结构信息
  - ✧ DNS(Domain Name System，域名系统)服务器
  - ✧网络地址块的分配和使用情况
  - ✧POC(Point of Contact，联系人)信息

✧例如：在Linux系统终端输入以下命令

whois 163.com

whois ustc.edu

Whois (Web)接口的链接地址

- http://www.internic.net
- 例如：查询以下Domain

  sohu.com; 163.com; foundstone.com

  (演示)

- **http://www.cnnic.net.cn/ 中国互联网络信息中心**
  (whois服务?)

  ustc.cn

whois 踩点对策

- 至少在最近一段时间内没有什么好办法能防止别人通过whois 查出企业网络的相关信息（如IP地址块）。但这并不等于说企业不能采取措施去干扰whois 或其他类似查询所检索出来的信息。

- 在向因特网注册机构提供企业信息的时候一定要谨慎从事，**不要把某位具体人员的直接联系方法或其他不应该暴露的信息填写出来**。特别是负责人的姓名、直拨电话和电子邮件地址不要填写到大型因特网注册机构的POC(point of contact，联系人)栏目里。

踩点方法二：
现代方法：利用因特网搜索引擎执行踩点

- 利用标准的搜索引擎在因特网上查找某个站点(或域)是件轻而易举的事。
  - 常用的搜索引擎是google(www.google.com.hk)和百度(www.baidu.com)，它能够给出非常详尽的信息。
- 利用搜索引擎可以查找到目标的Web主页，从而进一步定位到目标网络。
  - 比如，攻击者想要收集一个公司的域名信息，就可以直接在baidu、Google等搜索引擎中输入该公司的名字，搜索结果中的第一项往往就是该公司的Web站点。
  - **实例：**通过baidu查“中国科学技术大学”“台湾大学”的相关信息。

7.2.2 网络扫描

- 攻击者常常通过扫描来发现目标系统的漏洞，然后通过漏洞来侵入目标系统。从技术上看，漏洞的来源主要包括以下方面：
  - (1)软件或协议的设计瑕疵。在很多早期的软件和协议设计过程中，基本没有考虑安全方面的因素，从而存在很多安全方面的缺陷和漏洞。
  - (2)软件或协议实现中的弱点。即使设计得非常完美，在实现过程中仍可能引入漏洞。
  - (3)软件或协议本身的瑕疵。比如没有进行数据内容和大小检查、没有进行完整性检查等。
  - (4)系统和网络配置错误。
- 有两种常用的扫描策略：一种是主动式策略，另一种是被动式策略。

主动和被动扫描

- **主动式扫描是基于网络的**
  - 通过执行一些脚本文件模拟对系统进行攻击的行为并记录系统的反应，从中发现可能的漏洞。
  - 主动式扫描一般可以分成：活动主机探测、ICMP查询、Ping扫描、端口扫描、指定漏洞扫描、综合扫描等。
  - 扫描方式分成两大类：慢速扫描和乱序扫描。
    - 慢速扫描是指对非连续端口进行的、源地址不一致的、时间间隔长而没有规律的扫描；
    - 乱序扫描是指对连续端口进行的、源地址一致的、时间间隔短的扫描。
- **被动式扫描策略是基于主机的**
  - 对系统中不合理的设置、脆弱的口令以及其他同安全规则相抵触的对象进行检查。被动式扫描不会对系统造成破坏，而主动式扫描会对系统进行模拟攻击，可能造成破坏。

网络扫描技术

- 网络扫描技术包括Ping扫描（确定哪些主机正在活动）、端口扫描（确定有哪些开放服务）、操作系统辨识（确定目标主机的操作系统类型）和安全漏洞扫描（确定目标上存在着哪些可利用的安全漏洞）。
- 从攻击者信息收集的角度来看，攻击目标信息的收集尤为重要。这些信息可能包括远程机器上运行的各种TCP/UDP服务、操作系统版本／类型信息、应用程序版本／类型信息、系统存在的安全漏洞等。
- 主机信息的收集主要通过一些扫描工具，如端口扫描来完成。

端口扫描的过程

- 一般端口扫描的过程如图7-1所示。
- 攻击者向受害者发送探测数据包，目的主机通常会根据不同的探测数据包予以回应，攻击者通过分析受害者的回应获取大量有用的主机信息。

图7-1 端口扫描过程

常见的扫描类型

(1) TCP连接扫描：这是最基本的扫描方式。通过connect()系统调用向目的主机某端口（如80端口）发送完整的TCP连接请求。如果能够顺利完成三次握手过程，则此端口开放。这种方法比较容易被操作系统检测。

(2) TCP SYN扫描（半连接扫描）：向目的主机只发送TCP连接请求(SYN请求)，如返回SYN/ACK，则说明目的主机相应端口处于等待连接状态；如收到RST/ACK应答，则端口未开放。这种方法比完整连接扫描更隐蔽，操作系统一般不予记录。

(3) TCP FIN扫描：扫描工具向目的端口发送FIN请求，如端口关闭，按照RFC793的要求，应该返回RST包。此方法适用于UNIX系统主机，Windows系统(没有遵守RFC793)不受影响。

常见的扫描类型

(4) TCP ACK扫描：向目标端口发送ACK包，可以用来检测防火墙安装情况，了解防火墙类型等信息。

(5) TCP Null扫描：根据目标主机的响应，可以判断操作系统是Windows还是UNIX。

(6) TCP RPC扫描：用于UNIX系统，检查远程过程调用的端口以及对应的应用程序及其版本等信息。

(7) UDP扫描：向目标端口发送UDP包，如返回“ICMP PORT UNREACHABLE”，则端口关闭；否则端口开放。

(8) ICMP扫描：通过向目的主机发送ICMP探测包，分析应答包数据可以探测目的主机操作系统类型等信息。由于ICMP探测包的隐蔽性（和正常数据包类似），所以，一般的入侵检测系统难以发现。

7.2.3 常见的扫描工具

- 著名的扫描工具包括Nmap、PortScan等，知名的安全漏洞扫描工具包括开源的Nessus及一些商业漏洞扫描产品如ISS的Scanner系列产品。
- Nmap(network mapper)是网络扫描和嗅探工具包，
  - 其基本功能有三个：一是探测一组主机是否在线；二是扫描主机端口，嗅探所提供的网络服务；三是推断主机所用的操作系统。
  - Nmap可用于扫描有两个节点的LAN，直至500个节点以上的网络。Nmap还允许用户定制扫描技巧，可以深入探测UDP或者TCP端口，直至主机所使用的操作系统；还可以将所有探测结果记录到各种格式的日志中，供进一步分析操作。
- Nmap使用命令方式进行扫描，支持的扫描类型非常丰富。
  - 例如，执行SYN扫描时的命令为nmap -sS 192.168.1.0/24，-sS表明扫描类型，192.168.1.0/24表明扫描的目标网段。

Nmap实际效果

- i@NS:~/work$ nmap -sS 192.168.86.0/24
- The program 'nmap' is currently not installed. You can install it by typing:
- sudo apt-get install nmap
- i@NS:~/work$ sudo apt-get install nmap
- [sudo] password for i:
- Reading package lists... Done
- i@NS:~/work$ nmap -sS 192.168.86.0/24
- You requested a scan type which requires root privileges.
- QUITTING!
- i@NS:~/work$ sudo nmap -sS 192.168.86.0/24

  Starting Nmap 6.40 (http://nmap.org) at 2019-04-08 10:57 CST

  Nmap scan report for 192.168.86.102

  Host is up (0.00081s latency).

  Not shown: 970 closed ports

  PORT STATE SERVICE

  7/tcp open echo

7.3 缓冲区溢出攻击

- 缓冲区溢出(buffer overflow)攻击是利用缓冲区溢出漏洞所进行的攻击行动。

- 缓冲区溢出是一种非常普遍、非常危险的漏洞，在各种操作系统、应用软件中广泛存在。

- 攻击者利用缓冲区溢出攻击，可以导致程序运行失败、系统关机、重新启动等后果，精心设计的缓冲区溢出攻击甚至可以利用它执行非授权指令，甚至可以取得系统特权，进而进行各种非法操作。

7.3.1 缓冲区溢出的基本原理

- 缓冲区溢出的基本原理是攻击者通过**向目标程序的缓冲区写超出其长度的内容**，造成缓冲区的溢出，从而破坏程序的堆栈，使程序转而执行其他指令，以达到攻击的目的。造成缓冲区溢出的原因是程序中**没有仔细检查用户输入的参数**。
- 一个有漏洞的C程序：

```
void function(char * in)
{
  char buffer[128];
  strcpy(buffer, in);
}
```

  - 在这个简单的函数中，函数strcpy()将直接把输入的字符串in中的内容复制到buffer中。
  - 程序在进行函数调用时，首先按顺序将函数参数、返回地址、框架指针(ESP，EBP)、局部变量等数据压入堆栈，如图7-5所示。

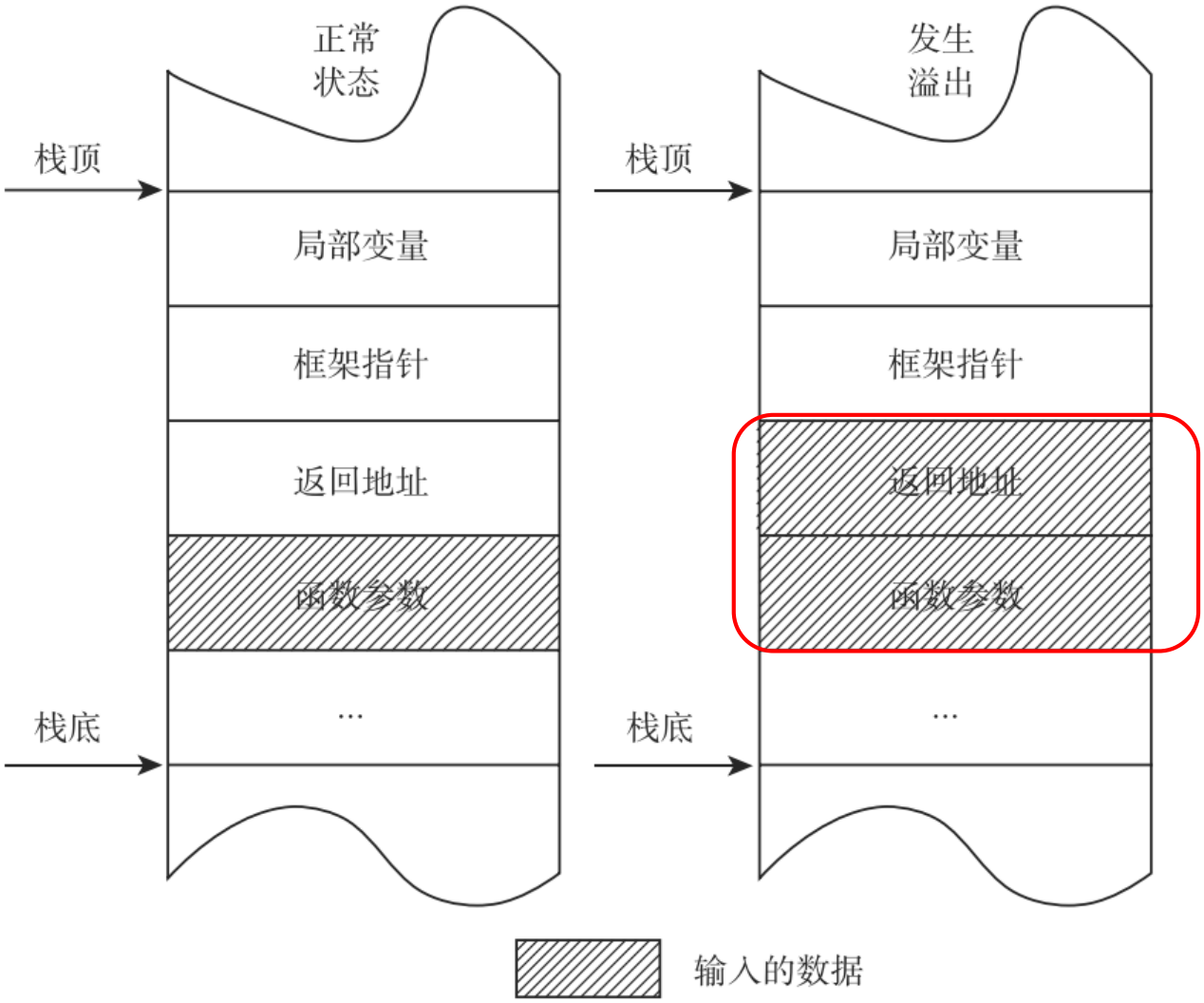

正常
状态
栈顶
局部变量
框架指针
返回地址
函数参数
…
栈底
发生
溢出
栈顶
局部变量
框架指针
返回地址
函数参数
…
栈底
输入的数据

图7-5 缓冲区溢出示意

溢出攻击的原理

- 如果输入in的长度大于128，就会造成缓冲区溢出，即输入数据**覆盖了邻近的缓冲区，有可能覆盖程序的正确返回地址**，造成程序运行出错。存在像strcpy这样的问题的标准函数还有strcat()、sprint()、vsprintf()、gets()、scanf()等。
- 通常情况下，攻击者往缓冲区中填过多的数据造成溢出只会出现**分段错误(segmentation fault)**，而不能达到控制目标主机的目的。
- 常见的攻击(漏洞利用)方法：
  - 通过制造缓冲区溢出，使程序转而**执行攻击者通过缓冲区溢出植入内存中的特殊指令**。
  - 如果该受到溢出攻击的程序有管理权限的话，攻击者可以很容易地获得一个有管理员权限的shell，从而实现对目标主机的控制。

缓冲区溢出漏洞

- 导致缓冲区溢出漏洞的原因：
  - 在程序设计过程中，未对输入数据的合法性（如长度）进行认真检查是导致缓冲区溢出存在的重要原因。
  - 数组访问越界、序列的下标(索引)越界，都是导致缓冲区溢出漏洞的原因。

- 缓冲区溢出漏洞在很多软件中都存在，根据计算机应急响应小组(CERT)的统计，超过50%的安全漏洞都是缓冲区溢出造成的。
- “红色代码”、“冲击波”、“Slammer蠕虫”等恶意代码均是利用不同的缓冲区溢出漏洞进行传播和实施攻击的。

7.3.2 缓冲区溢出的防范

- 对缓冲区溢出攻击的防范，目前有三种直接的保护法。
  - √(1)通过操作系统控制使接收转入数据的**缓冲区不可执行**，从而阻止攻击者植入攻击代码。
  - √(2)要求程序员**编写正确的代码**，包括严格检查数据、不使用存在溢出风险的函数、利用Fault injection等工具进行代码检查等。
  - √(3)利用**C编译器的边界检查**来实现缓冲区的保护，这个方法使得**有缓冲区溢出漏洞的进程不能被控制（劫持）**，但是相对而言代价比较大。

7.4 拒绝服务攻击

- 拒绝服务攻击即攻击者设法使目标主机停止提供服务，是常见的一种网络攻击手段。
- 其主要原理是利用网络协议的缺陷，采用耗尽目标主机的通信、存储或计算资源的方式来迫使目标主机暂停服务甚至导致系统崩溃。
- 拒绝服务攻击可以分为拒绝服务(denial of service，DoS)攻击和分布式拒绝服务(distributed denial of service，DDoS)攻击。

- 拒绝服务攻击容易引起目标的警觉，攻击者只有在其他攻击方式无效的情况下使用。

7.4.1 常见的拒绝服务攻击

1. SYN泛洪攻击

- SYN泛洪(flooding)攻击是目前流行的拒绝服务攻击之一，这种攻击利用TCP缺陷，发送大量伪造的TCP连接请求，**TCP连接无法完成第三步握手**，使被攻击主机的资源耗尽(CPU满负荷或内存不足)而停止服务。

- 其攻击过程如图7-6所示。

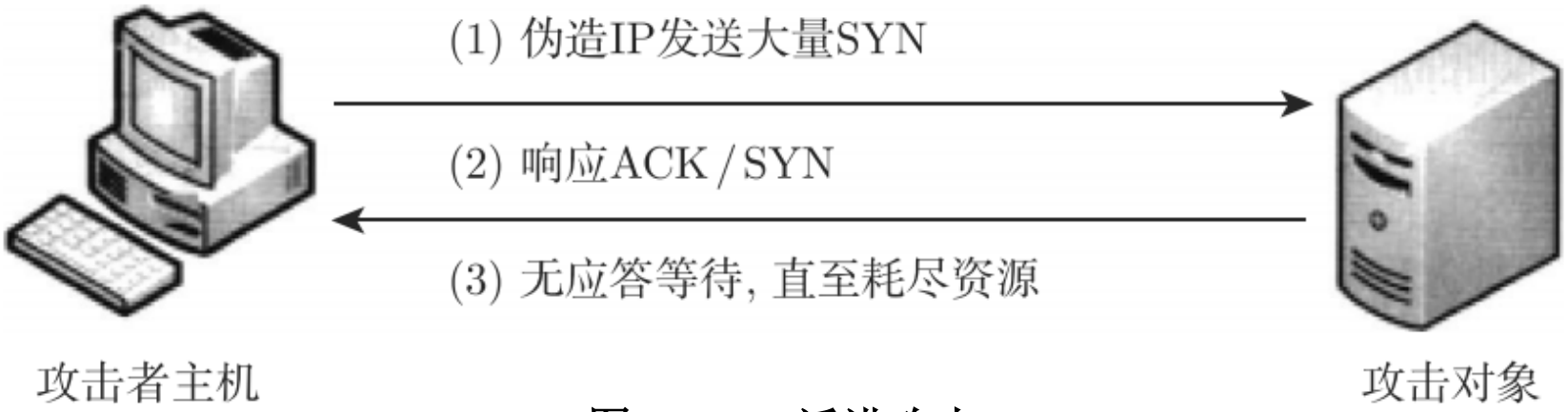


图7-6 SYN泛洪攻击

UDP泛洪攻击和Ping泛洪攻击

2. UDP泛洪攻击

- 攻击者利用简单的TCP/IP服务，如字符发生器协议(chargen)和Echo，来传送占满带宽的垃圾数据，通过伪造与某一主机的Chargen服务之间的一次UDP连接，回复地址指向开着Echo服务的一台主机。这样就在两台主机之间存在很多的无用数据流，这些无用数据流会导致针对带宽服务的攻击。

3. Ping泛洪攻击

- 由于在早期阶段，路由器对包的最大尺寸都有限制，许多操作系统对TCP/IP堆栈的实现在ICMP包上都是规定64kB，并且在对包的标题头进行读取之后，要根据该标题头里包含的信息来为有效载荷生成缓冲区。当产生畸形时，声称自己的尺寸超过ICMP上限的包，也就是加载的尺寸超过64kB上限时，就会出现内存分配错误，导致TCP/IP堆栈崩溃，致使接收方主机宕机。

泪滴攻击和Land攻击

4. 泪滴攻击

- 泪滴(teardrop)攻击是利用在TCP/IP堆栈中，实现信任IP碎片中的包的标题头所包含的信息来实现自己的攻击。IP分段含有指明该分段所包含的是原包的哪一段的信息，某些TCP/IP(包括Service Pack 4以前的Windows NT)在收到含有重叠偏移的伪造分段时将崩溃。

5. Land攻击

- Land攻击原理是设计一个特殊的SYN包，它的源地址和目标地址都被设置成某一个服务器地址。此举将导致接收服务器向它自己的地址发送SYN-ACK消息，结果这个地址又发回ACK消息并创建一个空连接。被攻击的服务器每接收一个这样的连接都将保留，直到超时。
- 不同的操作系统对Land攻击的反应不同，大多数UNIX系统将崩溃，Windows NT系统则变得极其缓慢（大约持续5分钟）。

6. Smurf攻击

- Smurf攻击的命名是因为第一个实现该类型攻击的软件名为Smurf，其原理是：
  - 通过向一个局域网的广播地址发出ICMP回应请求(echo request)，并将请求的返回地址设为被攻击的目标主机，导致目标主机被大量的应答包(echo reply)淹没，最终导致目标主机崩溃。

- 在这种攻击方式中，攻击者不直接向目标主机发送任何数据包，而是引导大量的数据包发往目的主机，因此也被称为“反弹攻击”，其攻击过程如图7-7所示。

Smurf攻击过程示意

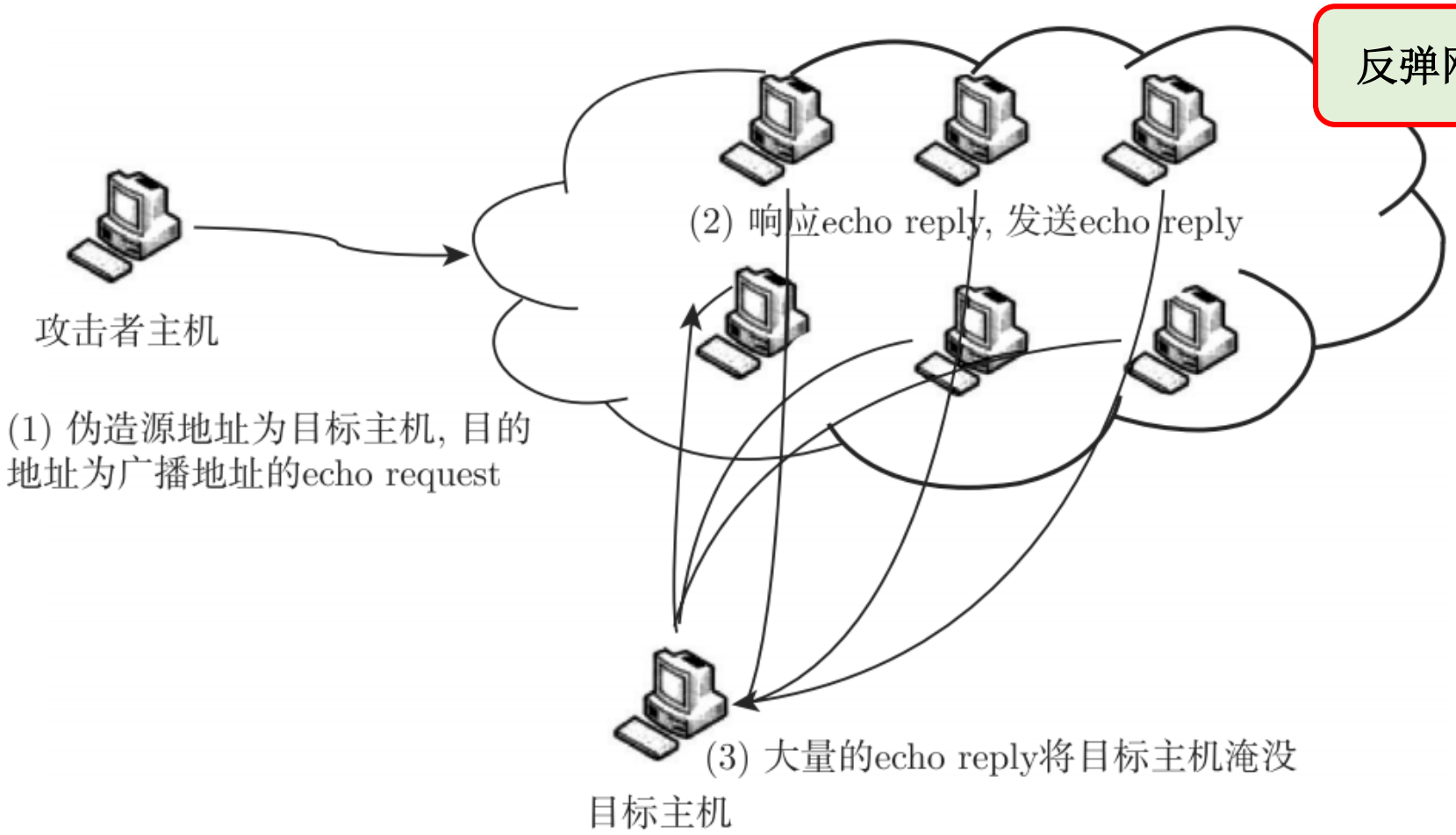


图7-7 Smurf攻击过程示意

7.4.2 分布式拒绝服务攻击

- 分布式拒绝服务(DDoS)攻击是在传统的DoS攻击方式上衍生出的新攻击手段。DDoS攻击指借助于客户 / 服务器技术，将多台主机联合起来作为攻击平台，对一个或多个目标发动DoS攻击，从而成倍地提高拒绝服务攻击的威力。
- 通常，攻击者使用一个主控程序控制预先被植入到大量傀儡主机中的代理程序。代理程序收到特定指令时就同时发动攻击，利用客户机 / 服务器技术，主控程序能在几秒钟内激活成百上千次代理程序的运行，因此能够产生比DoS攻击更大的危害后果。
- 其攻击过程如图7-8所示。

DDoS攻击示意

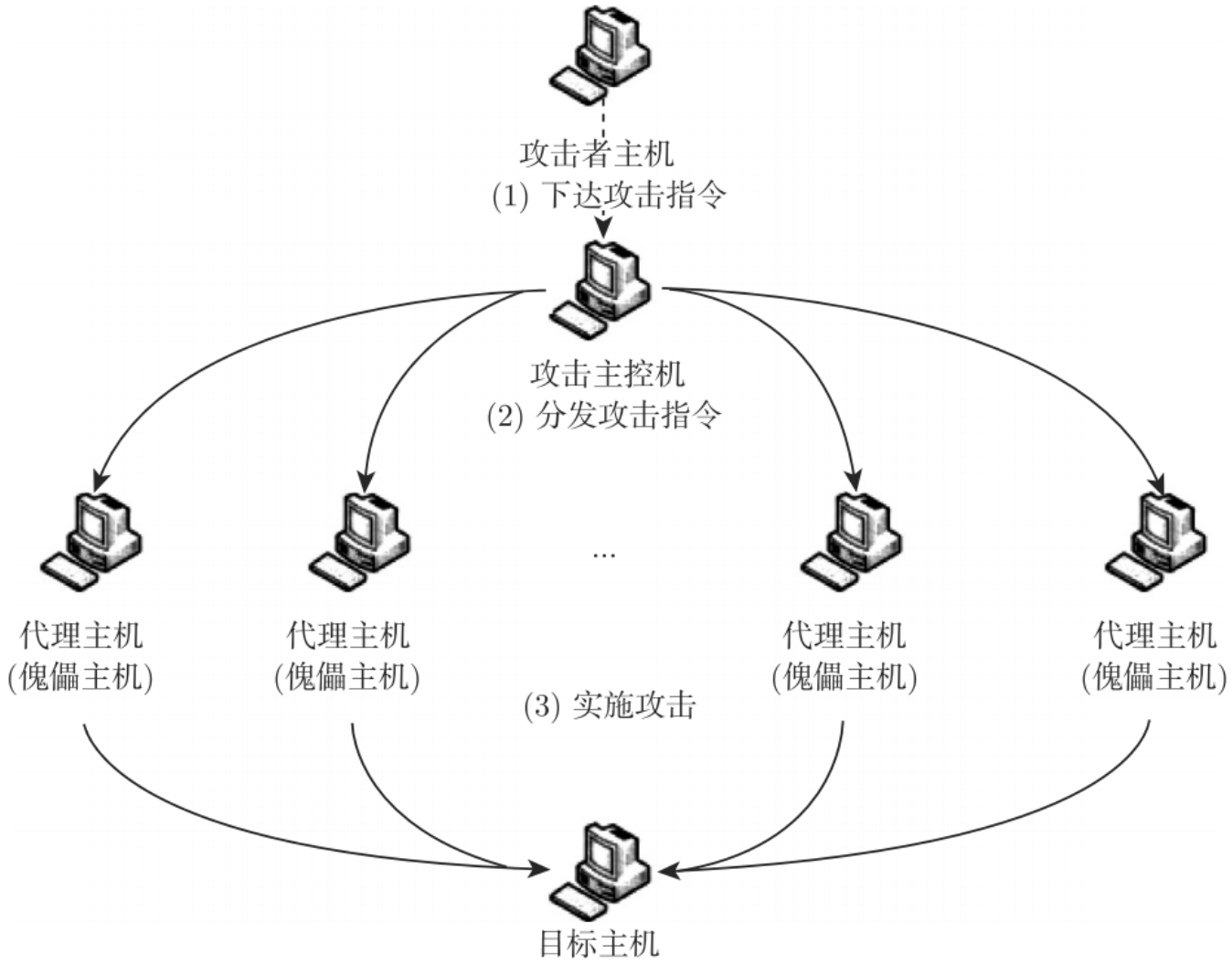


图7-8 DDoS攻击示意

7.4.3 拒绝服务攻击的防范

- 目前防范DoS/DDoS攻击的方法主要是从主机设置和网络设备设置两方面来考虑。

1. 主机设置

- 大多数主机的操作系统都提供了一些抵御DoS/DDoS的基本设置，主要包括以下内容：
  - ① 关闭不必要的服务。
  - ② 限制同时打开的SYN半连接数目。
  - ③ 缩短SYN半连接的超时等待(time out)时间。
  - ④ 除此之外，及时更新系统补丁也非常重要。
    - ➢例如，泪滴攻击主要是由于在早期操作系统的TCP/IP协议栈实现过程中，未对IP包进行合法性检查而直接处理造成的，经过升级后就能防范类似的攻击。

2. 网络设备设置

- 防火墙的设置主要包括以下方面：
  - (1)禁止对主机的非开放服务的访问。
  - (2)限制同时打开的SYN最大连接数。
  - (3)限制特定IP地址的访问。
  - (4)启用防火墙的DoS/DDoS的属性。
  - (5)严格限制对外开放的服务器的向外访问，这主要是防止服务器被攻击者利用。

路由器的设置

(1)使用扩展访问列表

- 扩展访问列表是防止DoS/DDoS攻击的有效工具，它既可以用来探测DoS/DDoS攻击的类型，也可以阻止DoS/DDoS攻击。
- 如Cisco路由器的命令：Show IP access-list，能够显示每个扩展访问列表的匹配数据包，根据数据包的类型，用户就可以确定DoS/DDoS攻击的种类。
- 如果网络中出现了大量建立TCP连接的请求，这表明网络受到了SYN Flood攻击，这时用户就可以改变访问列表的配置，阻止DoS/DDoS攻击。

(2)使用QoS

- 使用QoS特征，如加权公平队列(WFQ)、承诺访问速率(CAR)、一般流量整形(GTS)以及定制队列(CQ)等，都可以一定程度上阻止DoS/DDoS攻击。

(3)使用单一地址逆向转发

- 逆向转发(RPF)是路由器的一个输入功能，该功能用来检查路由器接口所接收的每一个数据包。如果路由器接收到一个源IP地址为10.10.10.1的数据包，但是路由表中没有为该IP地址提供任何路由信息，路由器就会丢弃该数据包。因此，逆向转发能够阻止Smurf攻击和其他基于IP地址伪装的攻击。

(4)使用TCP拦截

- Cisco公司的路由器在IOS 11.3版以后，引入了TCP拦截功能，这项功能可以有效防止SYN Flood攻击内部主机。

(5)使用基于内容的访问控制

- 基于内容的访问控制(CBAC)是对传统访问列表的扩展，它基于应用层会话信息，智能化地过滤TCP和UDP数据包，可以防止DoS/DDoS攻击。
- 由于DoS/DDoS攻击通常利用的是网络协议(如TCP/IP)中的缺陷，想要完全避免，则必须消除所有协议中的缺陷，在现阶段这种方式是不现实的。使用上述的方法并不能完全避免DoS/DDoS攻击，但能起到一定的缓解和预防作用。

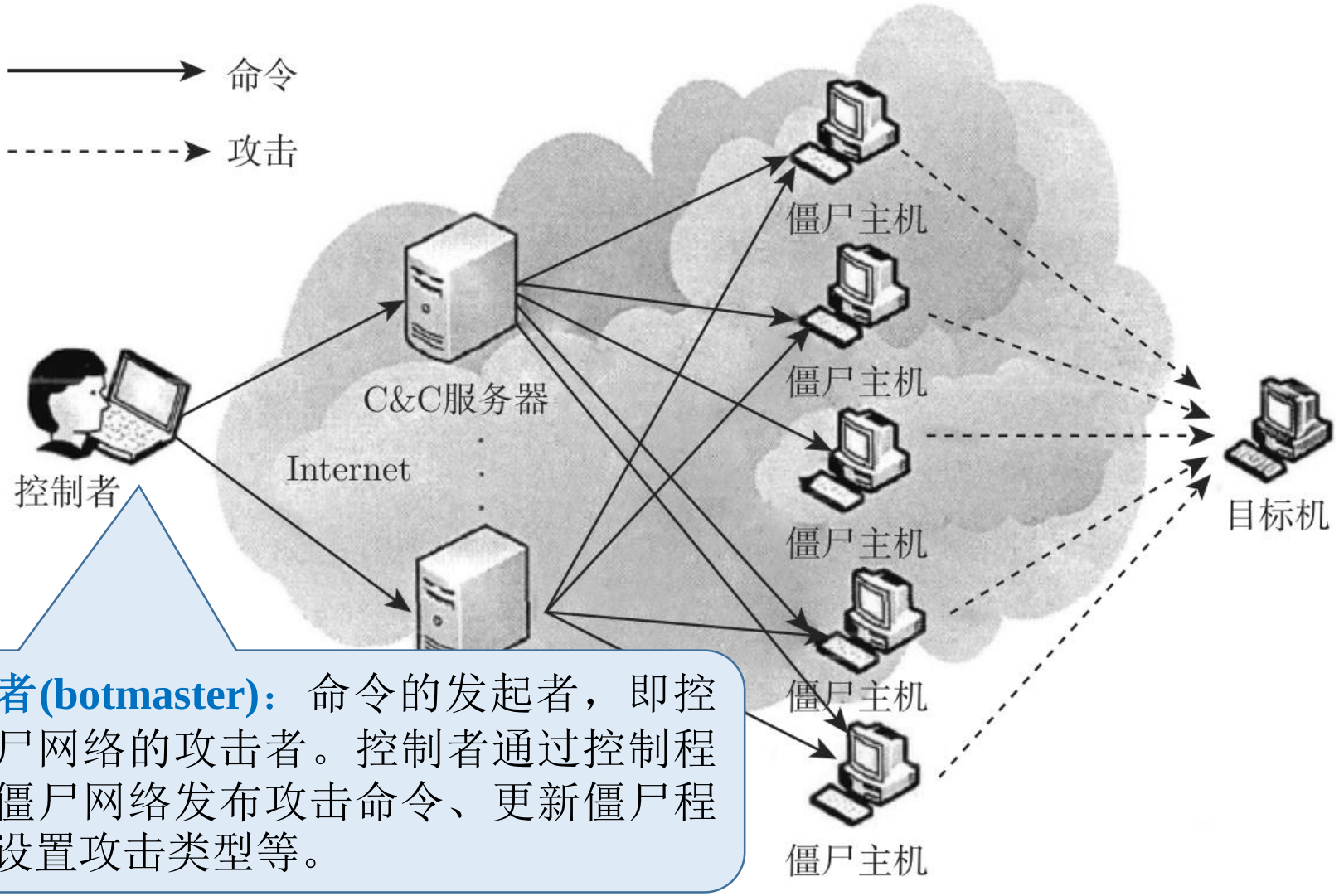
7.5 僵尸网络

- 近年来，**僵尸网络(BotNet)**逐渐发展成为攻击者手中最有效的攻击平台，是当前互联网面临的主要安全威胁之一。

1. 概念与结构

- 僵尸网络是攻击者出于恶意目的，融合传统的恶意软件，如计算机病毒、蠕虫和木马等技术，传播僵尸程序传染大量主机，并通过一对多的命令与控制(command and control，C&C)信道控制被感染的主机所组成的叠加网络(overlay network)。僵尸网络为攻击者提供了隐匿、灵活且高效的一对多命令与控制机制，是一个能发起多种大规模攻击的平台。利用僵尸网络，攻击者可以实现多种恶意活动，如分布式拒绝服务(DDoS)攻击、垃圾邮件、钓鱼网站、信息窃取等。

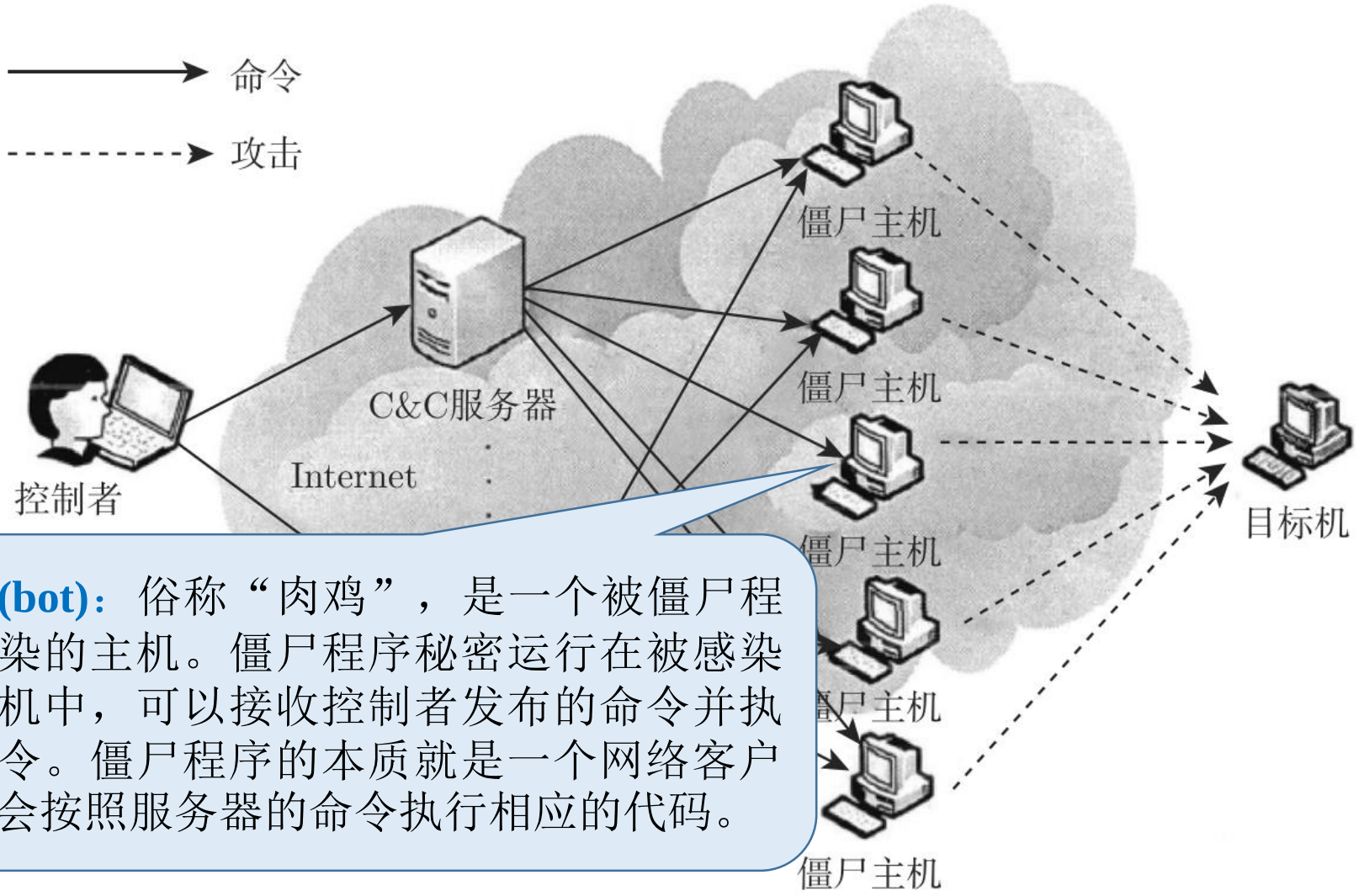
僵尸网络结构



控制者(botmaster)：命令的发起者，即控制僵尸网络的攻击者。控制者通过控制程序给僵尸网络发布攻击命令、更新僵尸程序、设置攻击类型等。

图7-9 僵尸网络结构

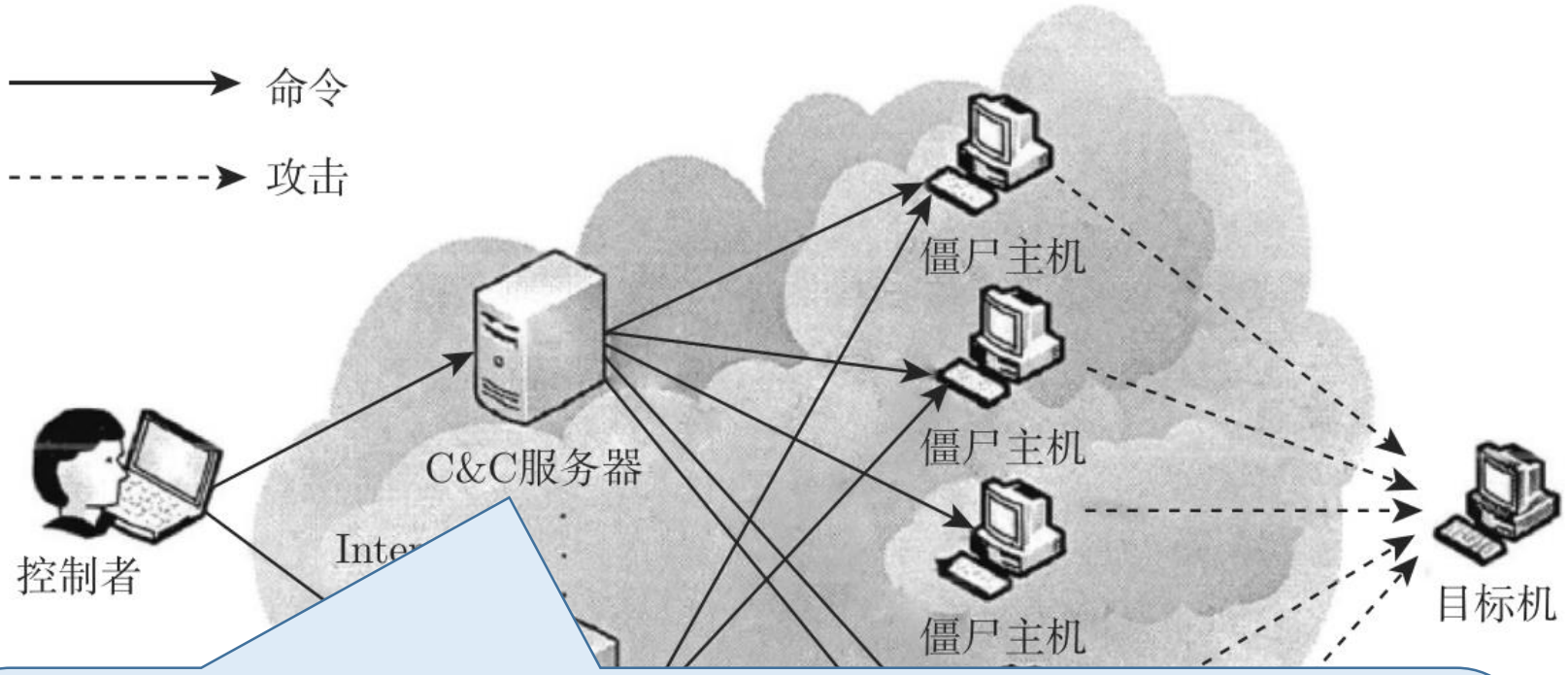
僵尸网络结构



主机(bot)：俗称“肉鸡”，是一个被僵尸程序感染的主机。僵尸程序秘密运行在被感染的主机中，可以接收控制者发布的命令并执行命令。僵尸程序的本质就是一个网络客户端，会按照服务器的命令执行相应的代码。

图7-9 僵尸网络结构

僵尸网络结构



命令与控制服务器(command and control server，C&C server)：控制者与僵尸主机通信的平台。控制者通过命令与控制服务器发布命令，僵尸主机则通过命令与控制服务器接收命令并向控制者发送命令执行报告。

命令与控制服务器通过专门的命令与控制(C&C)信道和僵尸主机进行通信。

僵尸程序的典型结构

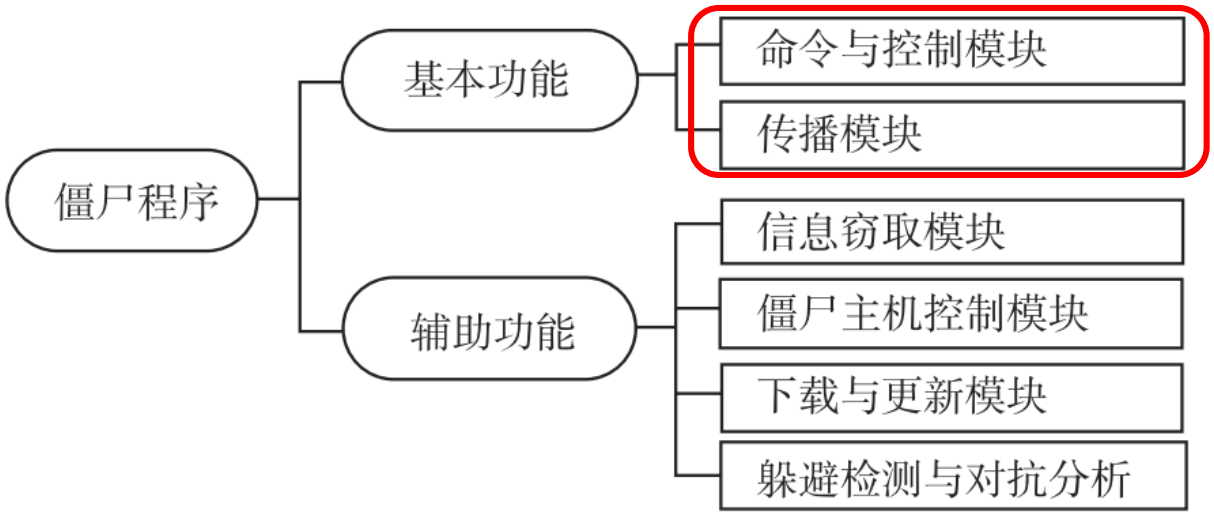


图7-10 僵尸程序功能结构

- **命令与控制**模块是整个僵尸程序的核心，实现了僵尸主机与控制器的交互，使僵尸主机接受攻击者的控制命令，进行解析和执行。
- **传播模块**通过各种不同的方式将僵尸程序传播到新的主机，使其加入僵尸网络接受攻击者的控制，从而扩展僵尸网络的规模。

僵尸程序的典型结构

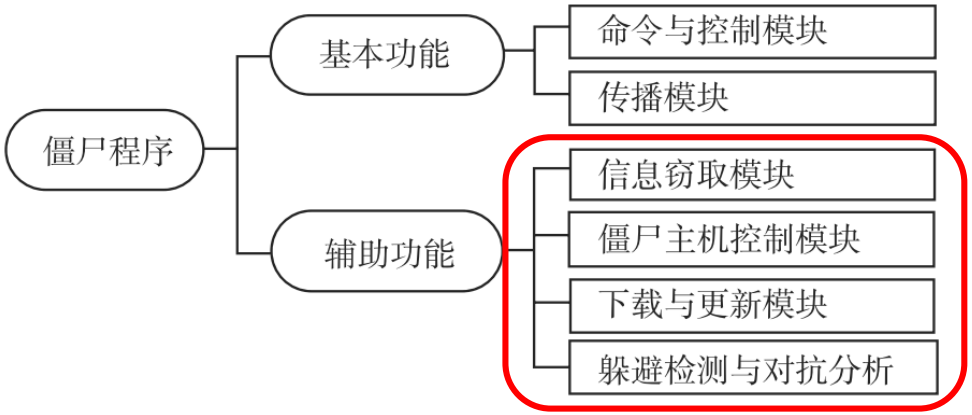


图7-10 僵尸程序功能结构

- **信息窃取**模块：用于获取感染主机信息（包括系统资源情况、进程列表、开启时间、网络带宽和速度情况等），以及搜索并窃取感染主机上有价值的敏感信息（如软件注册码、电子邮件列表、账号口令等）。
- **僵尸主机控制**模块：是攻击者利用受控的大量僵尸主机完成各种不同攻击目标的模块集合，包括DDoS攻击模块、发送垃圾邮件模块以及点击欺诈模块等。
- **下载与更新**模块：为攻击者提供向受控主机注入二次感染代码以及更新僵尸程序的功能。
- **躲避检测与对抗分析**模块：对僵尸程序进行多态、变形、加密等，并通过各种方式进行实体隐藏，以及检查debugger的存在、识别虚拟机环境、杀死反病毒进程、阻止反病毒软件升级等功能，使得僵尸程序能够躲避受控主机使用者和反病毒软件的检测。

2．工作机制

(1)感染目标主机，构建僵尸网络

- 攻击者会通过各种方式侵入主机，植入僵尸程序来构建僵尸网络。传播方式主要有远程漏洞攻击、弱口令扫描入侵、邮件附件、恶意文档、文件共享等。
- 早期的僵尸网络主要以类似蠕虫的主动扫描结合远程漏洞攻击进行传播，这种方式的主要弱点是不够隐蔽，容易被检测到。
- 近年来，僵尸网络逐渐以更为隐蔽的网页或邮件木马为主要传播方式，而且还加入了更多社会工程(social engineering)手段，更具有欺骗性。
  - 比如，根据近期的新闻和热点事件来变换其传播邮件的标题和内容，以增加其成功传播的几率。感染目标主机之后，攻击者会加载隐藏模块，通过变形技术、多态技术、rootkit技术等，使得僵尸程序隐藏于被控主机中。同时攻击者会加载通信模块，构建命令与控制信道，实现一对多的控制关系。

(2)发布命令，控制僵尸程序。

- 根据命令获取方式的不同，可以分为推模式(push)和拉模式(pull)。推模式是指僵尸主机在平时处于等待状态，僵尸控制程序主动向僵尸主机发送命令，僵尸主机只有被动接收到控制端命令后才进入下一步动作。拉模式是指控制端程序会将命令代码放置在特定位置，僵尸程序会定期从该位置主动去读取代码，作为进行下一步动作的命令。

(3)展开攻击。

- 根据攻击位置可将僵尸网络发起的攻击分为本地攻击和远程攻击。本地攻击指发起针对僵尸网络内部被控主机的攻击，比如对用户隐私信息的窃取等。远程攻击指攻击非僵尸网络内部的主机，根据目的不同又分为两类：一类是扩展僵尸网络规模，这类攻击目的是让外部主机感染同样的僵尸程序，最终成为僵尸网络的一部分。另一类攻击是打击、渗透方式，比如分布式拒绝服务、垃圾邮件等。

(4)攻击善后。主要目的是隐藏攻击痕迹，防止被追踪溯源。

7.6 缓冲区溢出漏洞的分析与利用

- 实验环境：32位Ubuntu Linux，版本16

http://mirrors.ustc.edu.cn/ubuntu-releases/16.04/ubuntu-16.04.6-desktop-i386.iso

- 关闭地址随机化：

  sudo sysctl -w kernel.randomize_va_space=0

7.6.1 缓冲区溢出攻击的原理

- 由于函数里局部变量的内存分配是发生在栈帧里的，所以如果在某一个**函数内部**定义了缓冲区变量，则这个缓冲区变量所占用的内存空间是在该函数被调用时所建立的栈帧里。
- 由于对缓冲区的潜在操作(比如字串的复制)都是从内存低址到高址的，而**内存中所保存的函数返回地址往往就在该缓冲区的上方(高地址)**——这是由于栈的特性决定的，这就为覆盖函数的返回地址提供了条件。
- 当用大于目标缓冲区大小的内容来填充缓冲区时，就可以**改写**保存在函数栈帧中的**返回地址**，从而改变程序的执行流程，**执行攻击者的代码**。
- 以下例程(attack_overflow.c)给出Linux IA32构架缓冲区溢出的实例。

IA32构架缓冲区溢出的实例
attack_overflow.c

```
// Define a large buffer with 32 bytes.
char Lbuffer[] = "01234567890123456789========ABCD";  //32Byte
// Define a large buffer with ATTACK_BUFF_LEN bytes.
#define ATTACK_BUFF_LEN 1024
char attackStr[ATTACK_BUFF_LEN];
void foo()
{
    char buff[16];
    strcpy (buff, attackStr);
}
void justCopyTheLbuffer()
{
    strcpy(attackStr, Lbuffer);
}
int main(int argc, char * argv[])
{
    justCopyTheLbuffer();  foo();    return 0;
}
```

注：蓝色字表示用户输入的信息

- 编译并运行该C程序：

```
$ gcc -fno-stack-protector -o buf attack_overflow.c
$ ./buf
Segmentation fault (core dumped)
$ gdb buf
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
(gdb) r
Starting program: /home/i/infosec/ch07/buf

Program received signal SIGSEGV, Segmentation fault.
0x44434241 in ?? ()
(gdb)
```

- 可见会发生段错误。

- 为了找出错误原因，需要用gdb对程序./buf进行调试。

  $ gdb buf

  ……

- 反汇编main和foo：

```
(gdb) disas main
Dump of assembler code for function main:
   0x08048534 <+0>:     lea    0x4(%esp),%ecx
   0x08048538 <+4>:     and    $0xfffffff0,%esp
   0x0804853b <+7>:     pushl  -0x4(%ecx)
   0x0804853e <+10>:    push   %ebp
   0x0804853f <+11>:    mov    %esp,%ebp
   0x08048541 <+13>:    push   %ecx
   0x08048542 <+14>:    sub    $0x4,%esp
   0x08048545 <+17>:    call   0x80484ad <justCopyTheLbuffer>
   0x0804854a <+22>:    call   0x804846b <foo>
   0x0804854f <+27>:    mov    $0x0,%eax
   0x08048554 <+32>:    add    $0x4,%esp
   0x08048557 <+35>:    pop    %ecx
   0x08048558 <+36>:    pop    %ebp
   0x08048559 <+37>:    lea    -0x4(%ecx),%esp
   0x0804855c <+40>:    ret
End of assembler dump.
(gdb)
```

```
(gdb) disas foo
Dump of assembler code for function foo:
   0x0804846b <+0>:     push   %ebp
   0x0804846c <+1>:     mov    %esp,%ebp
   0x0804846e <+3>:     sub    $0x18,%esp
   0x08048471 <+6>:     sub    $0x8,%esp
   0x08048474 <+9>:     push   $0x804a0a0
   0x08048479 <+14>:    lea    -0x18(%ebp),%eax
   0x0804847c <+17>:    push   %eax
   0x0804847d <+18>:    call   0x8048320 <strcpy@plt>
   0x08048482 <+23>:    add    $0x10,%esp
   0x08048485 <+26>:    nop
   0x08048486 <+27>:    leave
   0x08048487 <+28>:    ret
End of assembler dump.
```

设置断点

- 在函数foo的入口、对strcpy的调用、出口及其它需要重点分析的位置设置断点：

```
(gdb) b *(foo+0)
Breakpoint 1 at 0x804846b
(gdb) b *(foo+18)
Breakpoint 2 at 0x804847d
(gdb) b *(foo+28)
Breakpoint 3 at 0x8048487
(gdb) display/i $eip
```

运行程序并在断点处观察寄存器的值

```
(gdb) r
Starting program: /home/i/work/buf

Breakpoint 1, 0x0804846b in foo ()
1: x/i $pc
=> 0x804846b <foo>:      push   %ebp
(gdb) x/x $esp
0xbfffef2c:0x0804854f
(gdb) x/i 0x0804854f
   0x804854f <main+27>: mov    $0x0,%eax
```

- 函数入口处的堆栈指针esp指向的栈（地址为0xbfffef2c）保存了函数foo()返回到调用函数(main)的地址（0x0804854f），即“函数的返回地址”。

- 为了核实该结论，可以查看main的汇编代码：

```
(gdb) disas main
Dump of assembler code for function main:
   0x08048534 <+0>:    lea    0x4(%esp),%ecx
   0x08048538 <+4>:    and    $0xfffffff0,%esp
   0x0804853b <+7>:    pushl  -0x4(%ecx)
   0x0804853e <+10>:   push   %ebp
   0x0804853f <+11>:   mov    %esp,%ebp
   0x08048541 <+13>:   push   %ecx
   0x08048542 <+14>:   sub    $0x4,%esp
   0x08048545 <+17>:   call   0x80484ad <justCopyTheLbuffer>
   0x0804854a <+22>:   call   0x804846b <foo>
   0x0804854f <+27>:   mov    $0x0,%eax
   0x08048554 <+32>:   add    $0x4,%esp
   0x08048557 <+35>:   pop    %ecx
   0x08048558 <+36>:   pop    %ebp
   0x08048559 <+37>:   lea    -0x4(%ecx),%esp
   0x0804855c <+40>:   ret
End of assembler dump.
```

- 记录堆栈指针esp的值，在此以A标记：**A=$esp=0xbfffef2c**

继续执行到下一个断点

```
(gdb) c
Continuing.

Breakpoint 2, 0x0804847d in foo ()
1: x/i $pc
=> 0x804847d <foo+18>:call   0x8048320 <strcpy@plt>
(gdb)
```

- 查看执行strcpy(des, src)之前堆栈的内容。

  由于C语言默认将参数逆序推入堆栈，因此，src（全局变量attackStr的地址）先进栈（高地址），des（foo()中buff的首地址）后进栈（低地址）。

```
(gdb) x/x $esp
0xbfffef00:        0xbfffef10
(gdb)
0xbfffef04:        0x0804a0a0
(gdb) x/x 0x0804a0a0
0x804a0a0 <attackStr>:    0x33323130
(gdb)
```

- 可见， attackStr 的地址0x0804a0a0保存在地址为0xbfffef04的栈中，
- buff的首地址0xbfffef10保存在地址为0xbfffef00的栈中。

- 令**B= buff的首地址=0xbfffef10**，则buff的首地址与返回地址所在栈的距离=A-B= 0xbfffef2c - 0xbfffef10 =**0x1c=28**。

  (gdb) p 0xbfffef2c - 0xbfffef10
  $1 = 28
  (gdb) p/x 0xbfffef2c - 0xbfffef10
  $2 = 0x1c
  (gdb)

- 因此，如果attackStr的内容超过28字节，则将发生缓冲区溢出，并且返回地址被改写。attackStr的长度为32字节，其中最后的4个字节为“ABCD”

  (gdb) x/x 0x0804a0a0 + 0x1c
  0x804a0bc <attackStr+28>: 0x44434241
  (gdb)

- 因此，执行strcpy(des, src)之后，返回地址由原来的0x0804854f变为”ABCD”（0x44434241），即返回地址被改写。
- 继续执行到下一个断点：

  (gdb) **c**
  Continuing.
  Breakpoint 3, 0x08048487 in foo ()
  1: x/i $pc
  => 0x8048487 <foo+28>: ret

- 即将执行的指令为ret。执行ret等价于以下三条指令：

  eip的值=esp指针指向的堆栈内容

  跳转到eip执行指令

  esp=esp+4

  (gdb) **x/x $esp**

  0xbfffeffc: 0x44434241

- 可见，执行ret之前的堆栈的内容为“ABCD”，即0x44434241。
- 可以推断执行ret后将跳到地址0x44434241去执行。

- 继续单步执行下一条指令：

  (gdb) **si**

  0x44434241 in ?? ()

  1: x/i $pc

  => 0x44434241: <error: Cannot access memory at address 0x44434241>

  (gdb) **x/x $eip**

  0x44434241: Cannot access memory at address 0x44434241

  (gdb)

- 可见程序指针eip的值为0x44434241，而0x44434241是不可访问的地址，因此发生段错误。eip=0x44434241，正好是“ABCD”倒过来，这是由于IA32默认字节序为little_endian（小端字节序，低字节存放在低地址）。
- 通过修改attackStr的内容（将“ABCD”改成期望的地址），就可以设置需要的返回地址，从而可以将eip变为可以控制的地址，也就是说可以控制程序的执行流程。

调试重点

- 在漏洞函数的3个地方设置断点：
  - (1) 第一条汇编语句：在此记下函数的返回地址（A=esp的值）(动态变化)
  - (2) 调用strcpy对应的汇编语句：记下smallbuf的**起始地址=$esp=B** (动态变化)，与A相减可以得到产生缓冲区溢出所需的字节数**Offset=A-B**
  - (3) ret语句：查看esp的内容，确定被修改的返回地址。

7.6.2 缓冲区溢出攻击技术

- 为了实现缓冲区溢出攻击，需要向被攻击的缓冲区写入合适的内容。为此，攻击者必须精心构造攻击串，并根据被攻击缓冲区的大小将shellcode（攻击代码）放置在的适当位置。在此以strcpy为例，说明攻击串的构造方法。考虑如下函数：

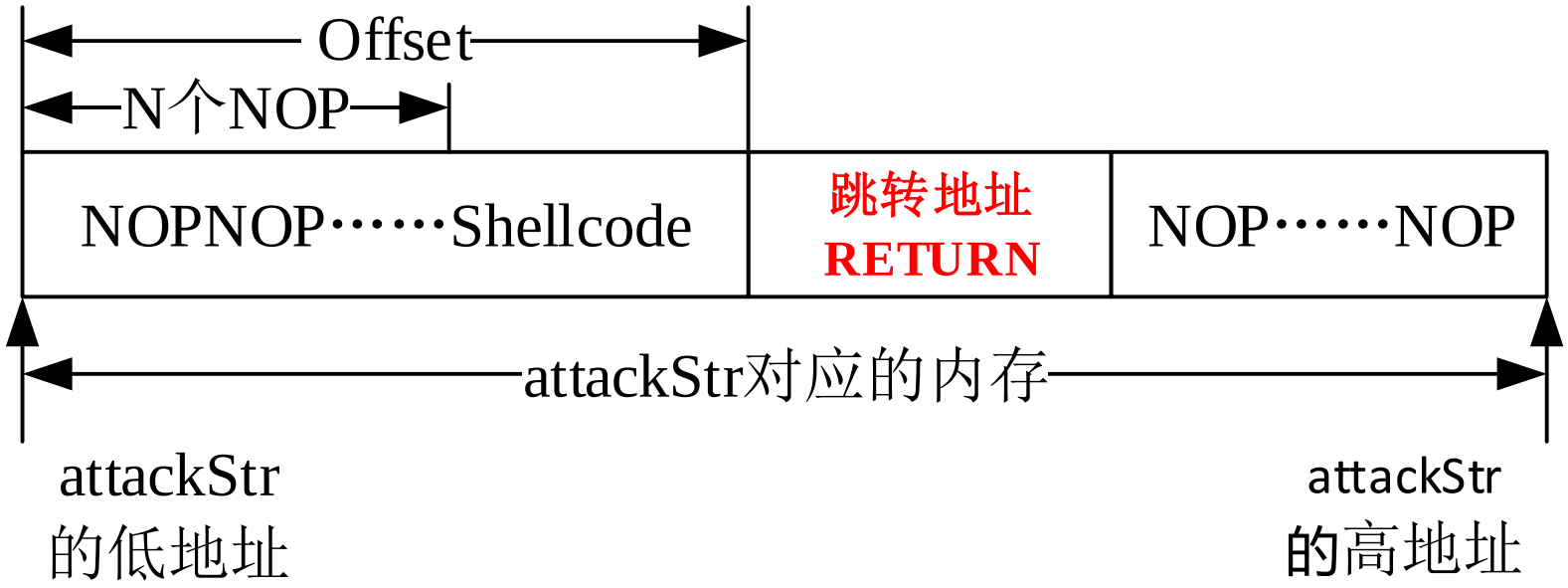
```
void foo()
{
    char buffer[LEN];
    strcpy (buffer, attackStr);
}
```

- 显然，若attackStr的内容过多，则上述代码会发生缓冲区溢出错误。在此buffer是被攻击的字符串，attackStr是攻击串。
- 假定attackStr是攻击者可以设置的，则有两种常用的方法构造attackStr。

方法一：将Shellcode放置在跳转地址(函数返回地址所在的栈)之前

- 如果被攻击的缓冲区(buffer)较大，足以容纳Shellcode，则可以采用这种方法。attackStr的内容按图7.6.2-1(a)的方式组织。
- 其中，Offset为被攻缓冲区(buffer)首地址与函数的返回地址所在栈地址的距离，需要通过gdb调试确定（见7.6.1）。
- 对于老版本的Linux系统，跳转地址RETURN的值可通过gdb调试目标进程而确定。然而，现代的操作系统由于在内核使用了地址随机化技术，堆栈的起始地址是动态变化的，进程每次启动时均与上一次不同，只能猜测一个可能的地址。

7.6.2-1(a) 攻击串的构造



7.6.2-1(b): 即将执行strcpy之前buffer及栈的内容

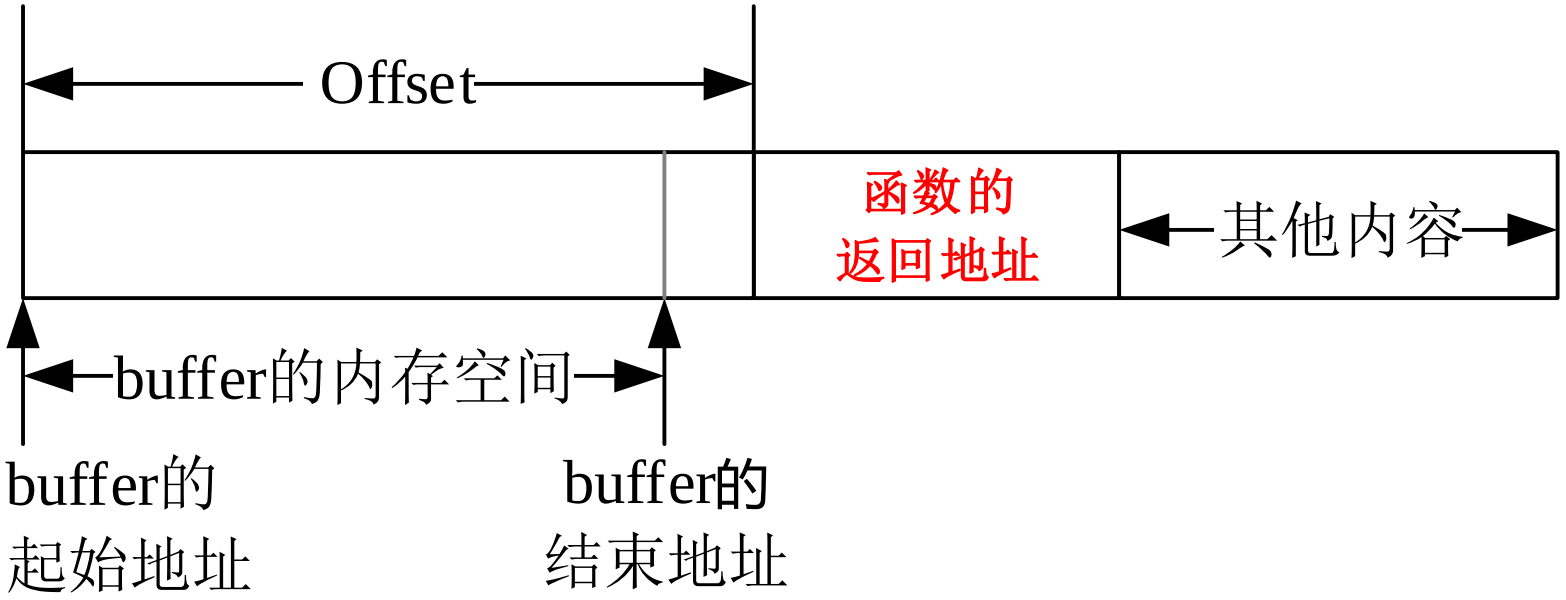


图7.6.2-2: 执行strcpy语句之后buffer及栈的内容

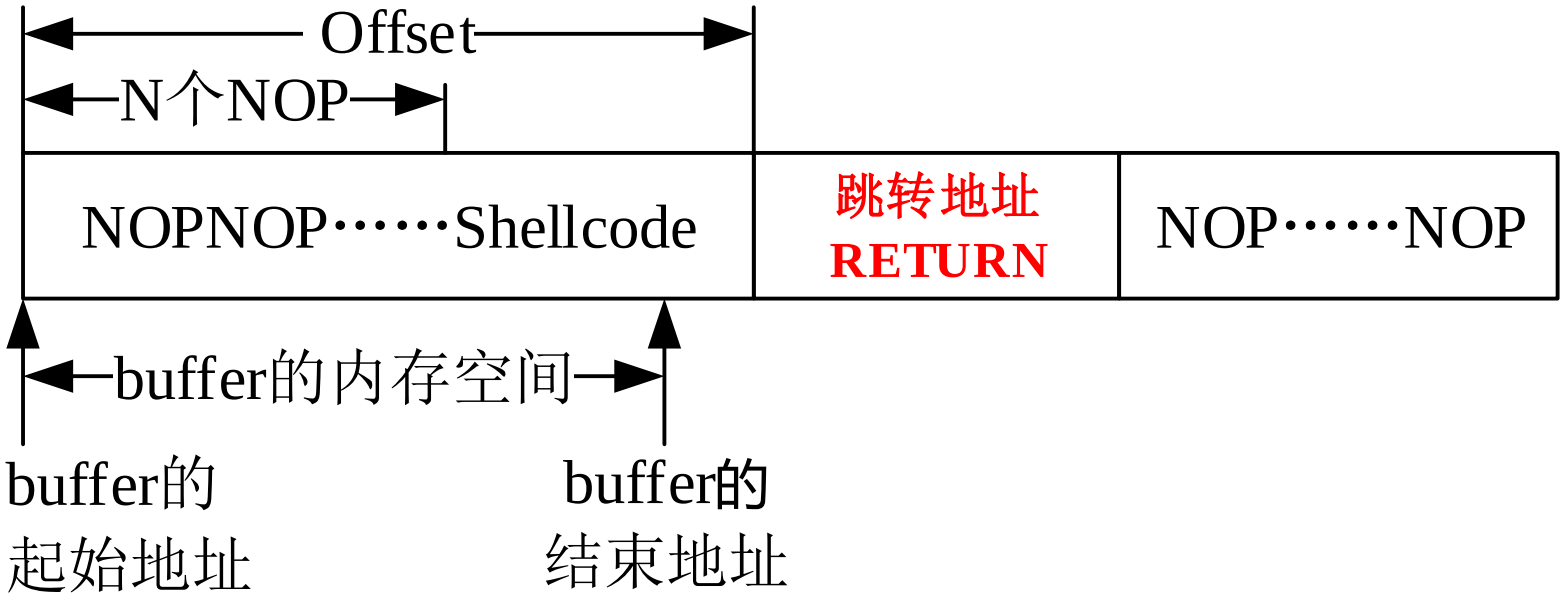


图7.6.2-1(a)中的跳转地址应按如下公式计算

RETURN=buffer的起始地址+n， 其中， 0<n<N

方法二：将Shellcode放置在跳转地址(函数返回地址所在的栈)之后

- 如果被攻击的缓冲区(buffer)的长度小于Shellcode的长度，不足以容纳shellcode，则只能将Shellcode放置在跳转地址之后。attackStr的内容按图7.6.2-3 (a)的方式组织。
- 即将执行strcpy (buffer, attackStr)语句时，buffer及栈的内容如图7.6.2-3(b)所示。执行strcpy (buffer, attackStr)语句之后，buffer及栈的内容如图7.6.2-3 -4所示。
- 图7.6.2-3 -3(a)中的跳转地址应按如下公式计算

  RETURN=buffer的起始地址+O**ffset+4**+n，

  其中， 0<n<N

图7.6.2-3 (a) 攻击串的构造

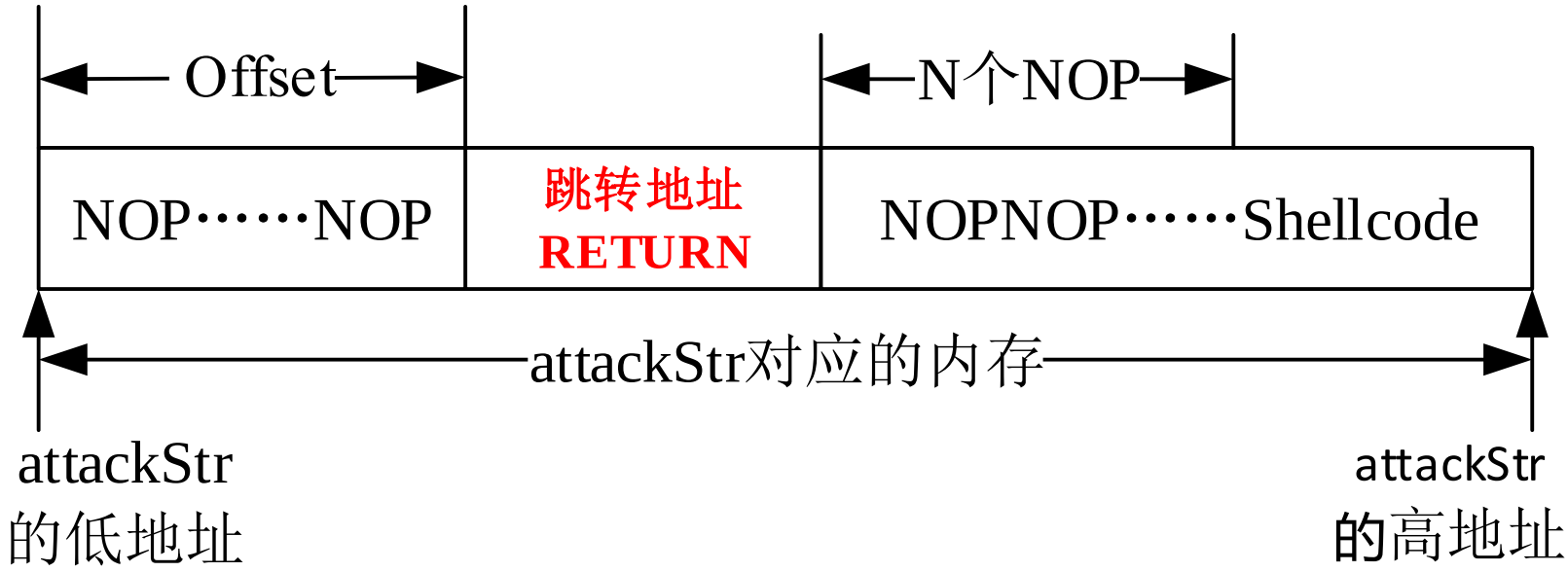


图7.6.2-3(b): 即将执行strcpy之前buffer及栈的内容

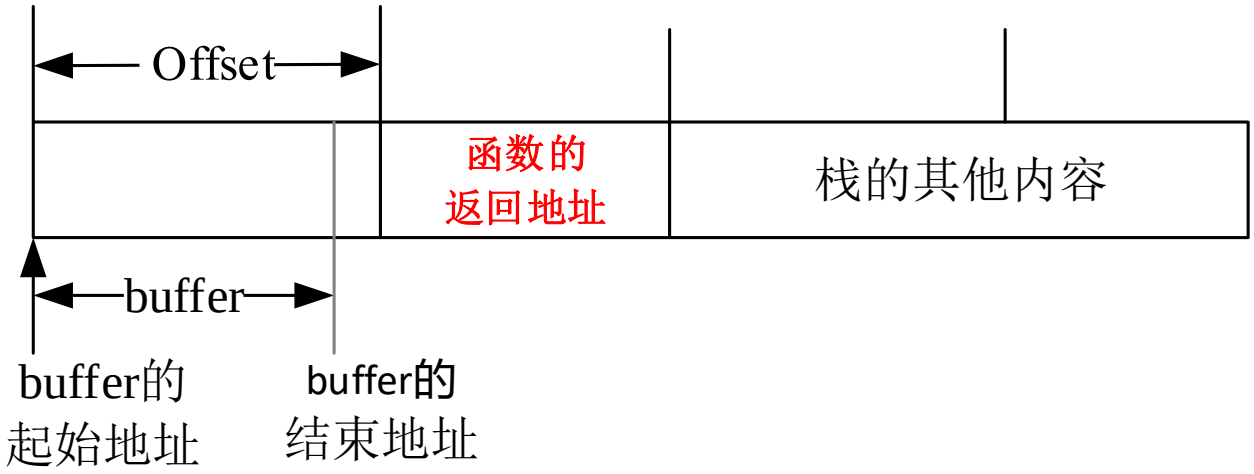
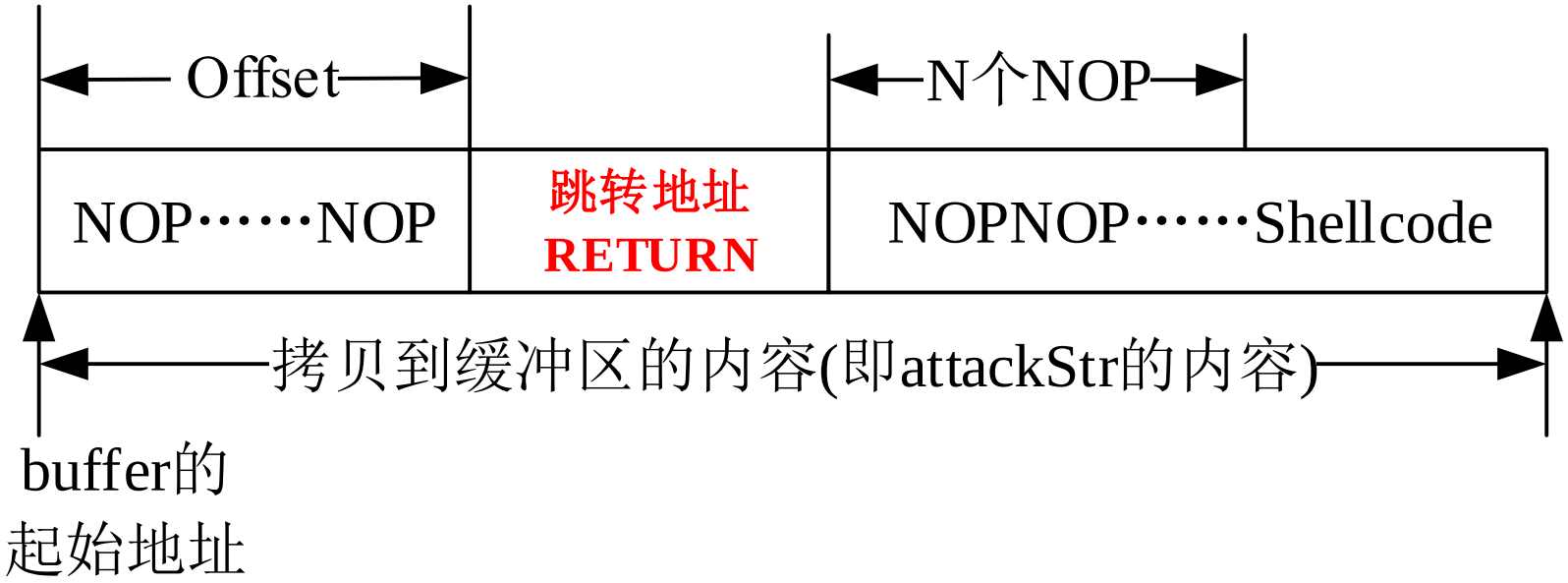


图7.6.2-4: 执行strcpy语句之后buffer及栈的内容



7.6.3 本地攻击实例

- 首先测试shellcode是否能成功运行。

```
int main(int argc, char * argv[])
{ run_shellcode();return 0; }
$ gcc -fno-stack-protector -o buf ../src/attack_overflow.c
$ ./buf
Segmentation fault (core dumped)
$ gcc -fno-stack-protector -z execstack -o buf ../src/attack_overflow.c
$ ./buf
$
```

- 组织攻击代码进行攻击

```
int main(int argc, char * argv[])
{ SmashSmallBuf(); foo(); return 0; }
```

见函数SmashSmallBuf()

拓展训练(自己练习，不考核)

(1) 32位Linux系统的缓冲区溢出攻击

- http://cybersecurity.ustc.edu.cn/ns/ns08.pdf
- http://cybersecurity.ustc.edu.cn/ns/ns08src2019-10.tar.gz

(2) Linux32 shellcode技术

- http://cybersecurity.ustc.edu.cn/ns/ns09.pdf
- http://cybersecurity.ustc.edu.cn/ns/ns09src2019-10.tar.gz

(3) 64位系统的缓冲区溢出攻击

- http://cybersecurity.ustc.edu.cn/ns/ns12.pdf
- http://cybersecurity.ustc.edu.cn/ns/ns12LinuxSrc2019-11.zip
- http://cybersecurity.ustc.edu.cn/ns/ns12Win64Src-2019-11.zip

(4) 攻击32位Windows7的缓冲区溢出漏洞

- https://mp.weixin.qq.com/s/n-6IJA4t2QZg1Mu5lP0zUw

谢谢！