第3章 SSL基础实验

SSL(Secure Sockets Layer,安全套接层)协议是网景公司提出的基于Web应用的安全协议。SSL协议指定了一种在应用层协议和可靠的传输层协议(如TCP协议)之间提供数据安全性分层的机制。它可以为网络连接提供数据加密、服务器认证、消息完整性以及可选的客户端认证。

本章要求是进行SSL相关实验,具体内容包括在VMware环境下实现SSL配置、SSL+FTP服务器搭建以及http与https通信数据分析对比。

3.1 SSL简介

SSL是一种在两台机器之间提供安全通道的协议,它具有保护传输数据以及识别通信机器的功能。安全通道是透明的,意思就是说它对传输的数据不加变更。客户与服务器之间的数据是经过加密的,一端读取的数据完全是另一端写入的内容。透明性使得几乎所有基于TCP的协议稍加改动就可以在SSL上运行,非常方便。

SSL原先是为Web设计的。网景公司的意图是针对其所有的通信安全问题,包括Web、电子邮件及新闻组通信提供一种单一的解决方案。SSL经过了多次修改,从开始的版本1到IETF最终所采纳的TLS标准。版本1从未经过广泛的部署,所以一般都认为从版本2开始。图3-1描述了各种SSL变种的谱系树。

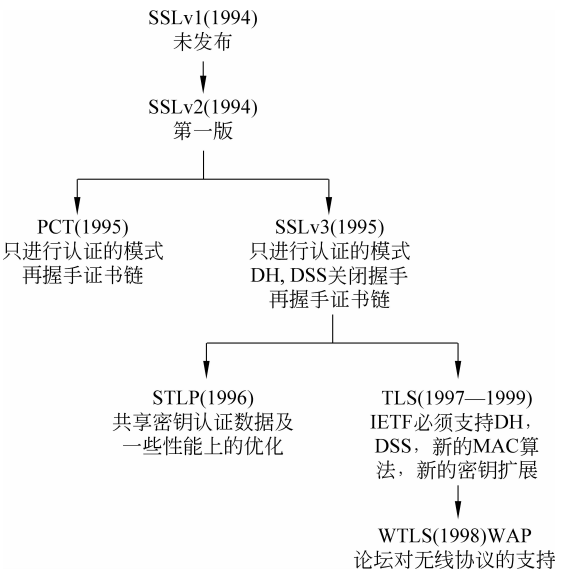

图3-1 SSL变种的谱系树

SSL 假定其下层的数据包发送机制是可靠的。写入网络的数据将依顺序发送给另一端的程序，不会出现丢包或重复发送的情况。从理论上讲，有许多传输协议都能提供此种服务，但在实际应用中，SSL 几乎只是在 TCP 上运行，它不能在 UDP 或直接在 IP 上运行。

3.1.1 SSL 协议结构

SSL 协议是一种分层协议，分为记录层协议和位于其上的握手层协议。握手层协议又可以细分为四种协议：握手协议（Handshake Protocol）、修改密钥参数协议（Change Cipher Spec Protocol）、警告协议（Alert Protocol）和应用数据协议（Application data Protocol）。握手层协议需要记录层协议的支持。图 3-2 描述了该协议的结构及其在 TCP/IP 协议栈中的位置。

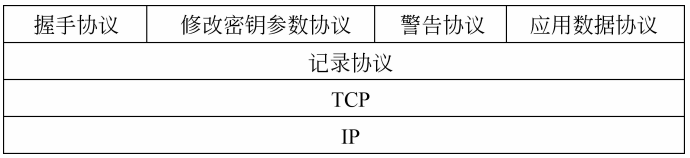

图 3-2　SSL 协议结构

SSL 连接分为两个阶段，即握手阶段和数据传输阶段。握手阶段对服务器进行认证并确立用于保护数据传输的加密密钥。握手完成之后，才能够开始传输数据。在传输阶段，数据被分成一系列经过保护的记录进行传输。

3.1.2 SSL 握手

SSL 握手有三个目的：第一，客户端与服务器端需要就一组用于保护数据的算法达成一致；第二，它们需要确立一组加密密钥；第三，握手还可以选择对客户端进行认证。在传输任何应用数据之前，都要使用握手协议。握手的整个工作过程如图 3-3 所示。

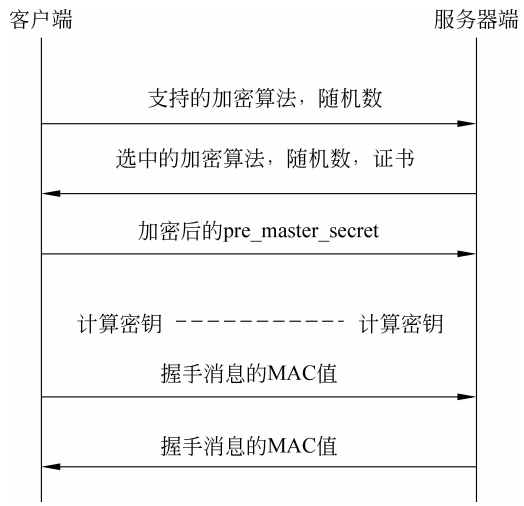

图 3-3　SSL 握手概述

(1) 客户端将它所支持的算法列表连同一个密钥产生过程用作输入的随机数发送给服务器。

(2) 服务器根据从列表的内容中选择一种加密算法,并将其连同一份包含服务器公钥的证书发回给客户端。该证书还包含了用于认证目的的服务器标识,服务器同时还提供了一个作为密钥产生过程部分输入的随机数。

(3) 客户端对服务器的证书进行验证,并抽取服务器的公钥。然后,再产生一个称作 pre_master_secret 的随机密码串,并使用服务器的公钥对其进行加密。最后,客户端将加密后的信息发送给服务器。

(4) 客户端与服务器端根据 pre_master_secret 以及客户端与服务器的随机数值独立计算出加密和 MAC(Message Authentication Code,消息认证码)密钥。

(5) 客户端将所有握手消息的 MAC 值发送给服务器。

(6) 服务器将所有握手消息的 MAC 值发送给客户端。

3.1.3　记录协议

握手的目的是建立起使发送和接收数据都受到保护的安全连接,而实际的数据传输需要使用记录协议来实现。

记录协议是通过将由高层获得的数据分割成一系列的片段并加以传输来工作的,其中对每个片段单独进行保护和传输。在接收方,对每条记录单独进行解密和验证。这种方案使得数据一经准备好就可以从连接的一端传送到另一端,接收端在接收到片段后可以立刻进行处理。

在传输片段前,先对每个片段进行压缩,然后通过计算压缩片段的 MAC 来保证其完整性,其中计算 MAC 使用的散列算法由握手过程协商的结果得到。MAC 与压缩片段一起进行传输,并由接收端加以验证。将 MAC 附加到压缩片段的尾部,然后对整个压缩片段和 MAC 进行加密,得到相应的负载。最后给负载加上头信息,头信息与负载一起就成为一条记录。

记录层协议工作流程如图 3-4 所示。详细步骤如下:

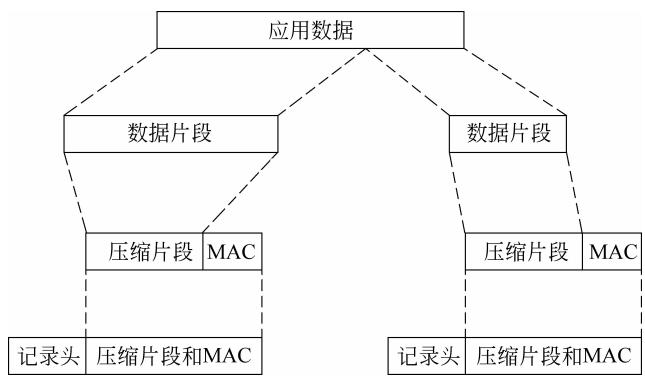

图 3-4　SSL 记录协议的工作流程

(1) 将上层数据分成适当大小的块。每个块的大小不得超过 2^{14}B(16 384B)。

(2) 对数据块进行压缩,压缩不能丢失信息,并且增加的长度不能超过 1024 字节。

(3) 在压缩后的数据上计算 MAC。此时需要使用共享的密钥。

(4) 使用同步加密算法对压缩报文加上 MAC 进行加密。加密对内容长度的增长不得超过 1024 字节,因此总长度不得超过 214+2048 字节。

(5) 对于流加密,压缩报文和 MAC 一起被加密;对于分组加密,在 MAC 之后,加密之前可以增加填充(padding)。填充由表示填充长度的字节跟着一定数目的填充字节组成,填充字节的数目是使得要加密的数据(明文加上 MAC,再加上填充)的总长度成为加密分组长度整数倍的最小数目。例如,一个 58 字节的明文(如果使用压缩,就是压缩正文),带有 20 字节的 MAC,使用分组长度为 8 字节的算法进行加密(如 DES)。加上填充长度字节,产生的总长度为 79 字节。为了使得总长度为 8 的整数倍,需要增加一个字节的填充。

(6) SSL 记录协议处理的最后一个步骤是附加一个首部。

3.1.4 SSL 警告和修改密钥参数协议

警告协议允许连接的一端向另一端报告警告信息。警告信息包括警告内容和严重级别。如果收到 fatal 级别的警告消息,则当前连接立即中断,并清除包含会话标识、密钥在内的所有状态参数。在这种情况下,与该会话相连的其他连接可以继续进行,但不能再重用该会话建立新的连接。同其他的消息一样,警告信息也要经过压缩和加密后再进行传输。警告消息的类型可分为关闭警告(close alert)和错误警告(error alert)。

修改密钥参数协议用来标识信号的转换,它只包含一条信息,信息内容为一个字节,其值为 1。修改密钥参数消息是由客户方或服务器发出,用来通知对方随后的记录将受到刚协商的密钥参数和密钥的保护。从握手协议的流程图可看出,客户方在发送完密钥交换和证书验证消息后发送修改密钥参数消息,服务器在成功处理了从客户方接收到的密钥交换消息以后也发送一个修改密钥参数消息。如果在握手过程中采用了会话重用,则双方在 hello 消息之后发送修改密钥参数消息。

3.2 SSL 配置实验

【实验目的】

(1) 掌握对 Web 数据进行 SSL 安全传输的技术;

(2) 掌握证书申请的过程;

(3) 熟悉 X.509 v3 证书格式。

【实验内容】

配置基于 Web 的 SSL 连接:

(1) IIS 服务器的启动;

(2) 证书服务的启动和独立根 CA 的安装;

(3) 服务器证书申请、颁发和安装;

（4）客户端证书申请、颁发和安装；

（5）在服务器上配置 SSL；

（6）客户端通过 SSL 与服务器建立连接；

（7）WireShark 抓包，分析 SSL 记录，并绘制 SSL 握手的流程。

【实验环境】

安装 Windows XP 的计算机、安装 Windows Server 2003 的计算机、IIS 的 Web 服务器和证书服务、局域网。

【实验参考步骤】

本实验主要讲述 SSL 配置、基于证书的身份认证和 CA 服务器搭建，即使用证书在 Web 服务器上设置 SSL。

（1）在服务器上安装 IIS 组件，并配置 Web 站点，如图 3-5 至图 3-7 所示。

图 3-5　安装 IIS

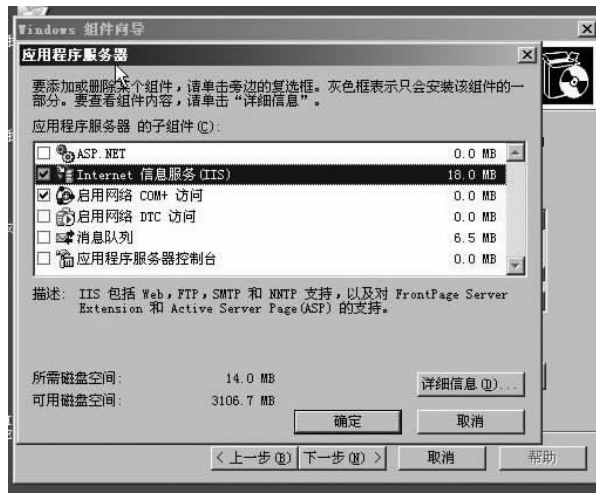

图 3-6　IIS 配置

图 3-7　配置 Web 服务器

(2) 使用 IE 浏览器输入 IP 地址访问本地站点,如图 3-8 所示。

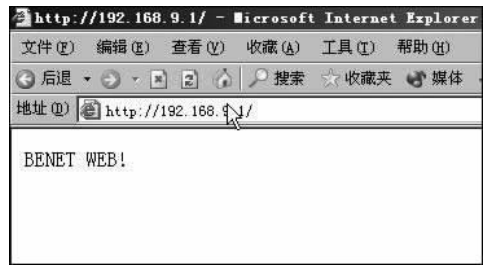

图 3-8　http 访问站点

(3) 在服务器上安装 CA 组件,如图 3-9 所示。

图 3-9　CA 安装

(4) 因为没有 AD(活动目录),所以只能创建独立 CA,又因为是第一个 CA,所以选择独立根 CA,并选择自定义安装,如图 3-10 至图 3-13 所示。

图 3-10　设置 CA 名字

图 3-11　选择证书数据库及其日志信息的位置

图 3-12　安装证书时需要停止 Internet 信息服务

图 3-13　启用 ASP 支持以完成安装

(5) 打开证书颁发机构查看，如图 3-14 所示。

图 3-14　证书颁发机构

(6) 使用 IIS 查看默认站点关于 CA 证书的列表，如图 3-15 所示。

图 3-15　CA 证书列表

(7) 在 Web 服务器上设置 SSL,生成证书申请,如图 3-16 所示。

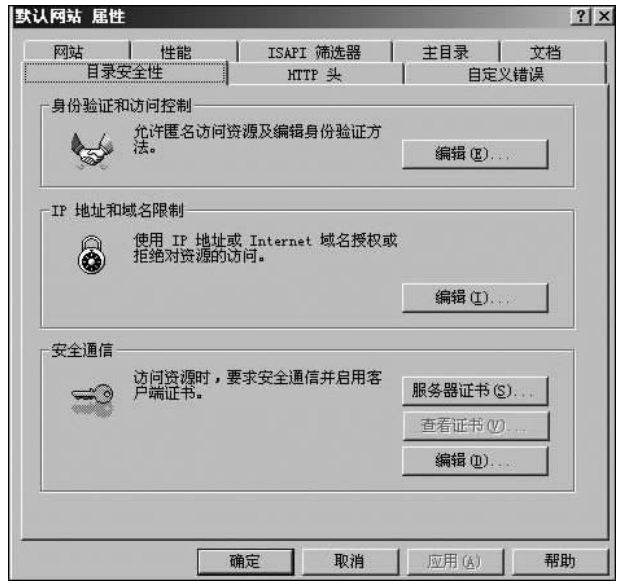

图 3-16　服务器配置 SSL

(8) 单击“安全通信”选项区域中的“服务器证书”按钮安装证书,选择新证书,如图 3-17 所示。

图 3-17　服务器证书申请

证书向导过程不一一赘述,填补完信息即可。

首先提交证书申请。

(1) 使用 IE 浏览器打开本地 Web 站点进入证书申请页面,如图 3-18 和图 3-19 所示。

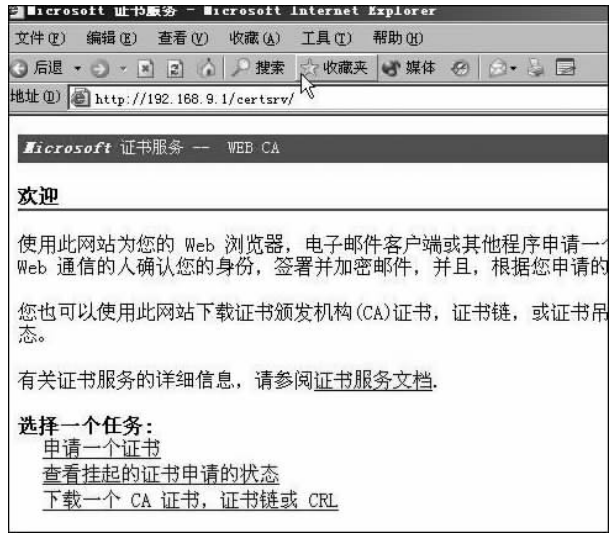

图 3-18　选择申请证书

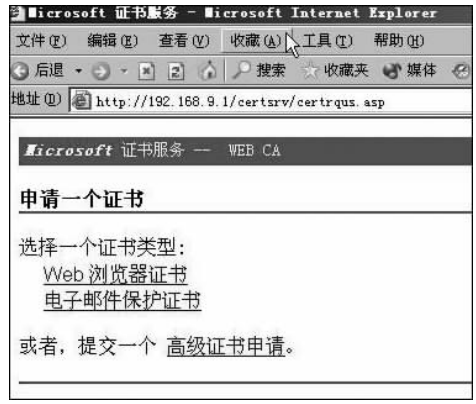

图 3-19　选择高级证书申请

（2）在提交一个证书申请中选择后者“BASE64 编码的证书申请”，如图 3-20 所示。

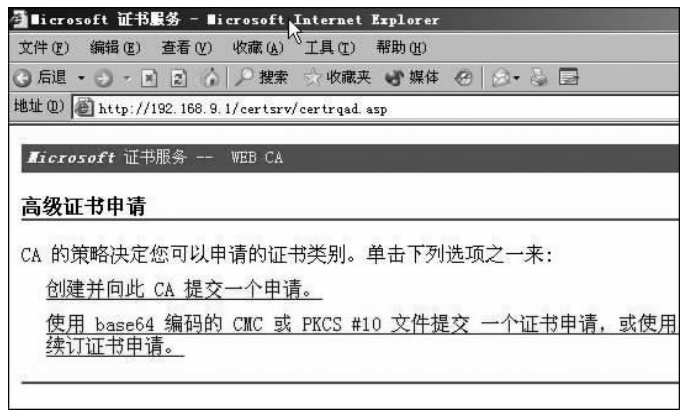

图 3-20　BASE64 编码的证书申请

（3）此时弹出文本框，需要输入内容，如图 3-21 所示。

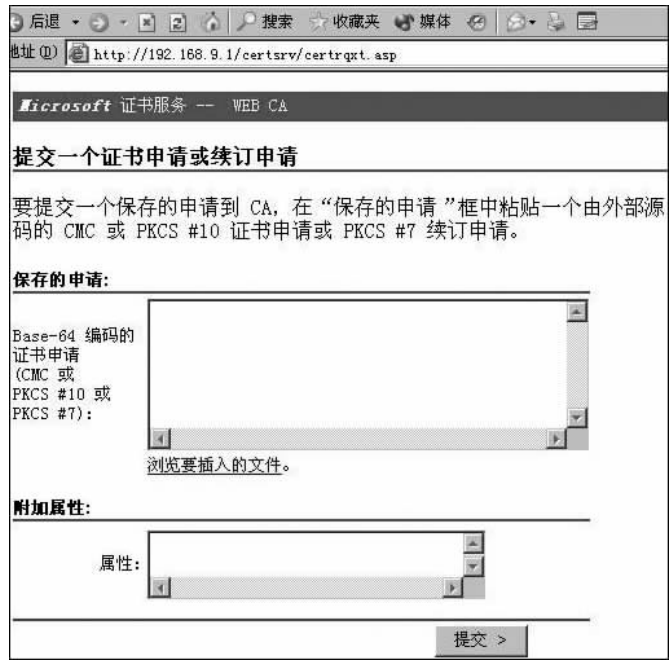

图 3-21　保存证书申请文件

（4）找到之前保存的 certreq.txt，打开并把其内容全部复制到文本框中，然后单击“提交”按钮，完成以上步骤后证书被挂起，如图 3-22 所示。

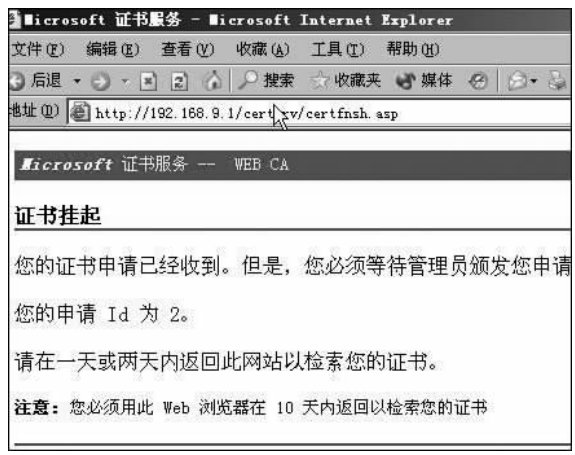

图 3-22　证书挂起

当证书申请被提交到证书颁发机构，下一步就要颁发证书了。

（1）打开证书颁发机构工具，选择“挂起的申请”选项，如图 3-23 所示，可以看到刚才申请后被挂起的证书，然后选中挂起的证书，右击选择“颁发”命令。

（2）使用 Web 形式访问证书申请页面，并选择下载证书，如图 3-24 所示。

图 3-23 颁发证书

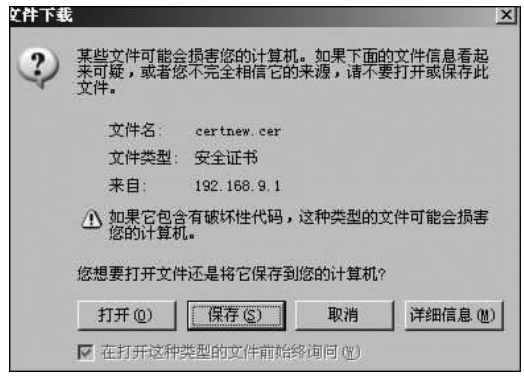

图 3-24 下载证书

(3) 在 Web 服务器上安装证书。

再次打开 IIS 服务器找到站点属性,再次单击“服务器证书”按钮,之后选择处理被挂起的请求,如图 3-25 所示。

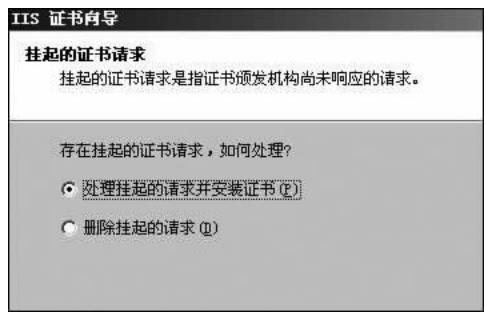

图 3-25 在 Web 服务器上安装证书

输入包含 CA 响应的文件的路径和文件名。

(4) 启用安全通道(SSL)。

打开站点属性,选择“安全通信”→“编辑”按钮,然后选中“要求安全通道”复选框即可,如图 3-26 所示。

(5) 使用 HTTPS 方式访问站点,如图 3-27 所示。

图 3-26　启用安全通道(SSL)

图 3-27　HTTPS 访问站点

【实验报告】

(1) SSL 协议的原理及运行机制；

(2) 基于 Web 的 SSL 连接的配置；

(3) 基于证书的身份证和 CA 服务器的搭建；

(4) WireShark 工具抓包分析 SSL 记录，并绘制 SSL 握手的流程。

3.3　小结

本章进行了 SSL 相关实验，对 SSL 协议的原理和结构有一定的认识和了解。通过 SSL 的配置实验，熟练掌握了对 Web 数据进行 SSL 安全传输的技术；同时熟悉了 X.509 v3 证书的格式以及申请证书的过程。

参 考 文 献

[1] 王志海等. OpenSSL 与网络信息安全：基础、结构和指令[M]. 北京：清华大学出版社，2007.

[2] 马小婷. 深入浅出密码学：常用加密技术原理与应用[M]. 北京：清华大学出版社，2012.

[3] 孙建国等. 网络安全实验教程[M]. 北京：清华大学出版社，2011.

[4] J. Frahim and Q. Huang. SSL Remote AccessVPNs [M]. 北京：人民邮电出版社，2009.

[5] Viega，John，Messier 等. Network Security with Open SSL [M]. O'Reilly Media，2002.

[6] Oppliger，Rolf. SSL and TLS：Theory and Practice [M]. Artech House Publishers，2009.

[7] Rescorla，Eric. SSL and TLS：Designing and Building Secure Systems [M]. Addison-Wesley Professional，2000.

[8] 周敬利，曾海鹏. SSL VPN 服务器关键技术研究[J]. 计算机工程与科学，2005(6).

第 4 章 缓冲区溢出攻击初级实验

在现有的攻击类型中，缓冲区溢出攻击是所有攻击类型中最为常见的一种。在所有的漏洞中，缓冲区溢出漏洞占了远程网络攻击中的绝大多数。缓冲区溢出攻击之所以成为一种常见的攻击手段，其原因在于缓冲区溢出漏洞容易实现并且最为普遍，另外，攻击者通过在远程靶机中植入并执行攻击代码，从而获取被攻击主机的权限，并以此做自己想做的任何事情。

目前的缓冲区攻击方式有很多种，其中最常见的有栈溢出、堆溢出、整型溢出、格式化字符串溢出及文件流溢出等。这些溢出攻击对原有程序具有很大的危害，其中包括应用程序异常，系统不稳定甚至崩溃，甚至程序跳转到恶意代码，控制权被窃取。本实验通过使用 OllyDbg 调试 OD 漏洞，理解缓冲区溢出攻击原理，了解栈溢出的攻击过程，从而在实际的写程序中尽量避免缓冲区溢出漏洞。

4.1 栈溢出原理

4.1.1 预备知识

栈是一段连续的内存，程序可以将数据压入栈中，也可以将数据从栈顶弹出，栈顶由称为 esp 的寄存器进行定位。压栈的操作使得栈顶的地址减小，弹出的操作使得栈顶的地址增大。其生长方向与内存的生长方向正好相反，从高地址向低地址生长，并且每一个线程有自己的栈。图 4-1 为内存分布示意图。

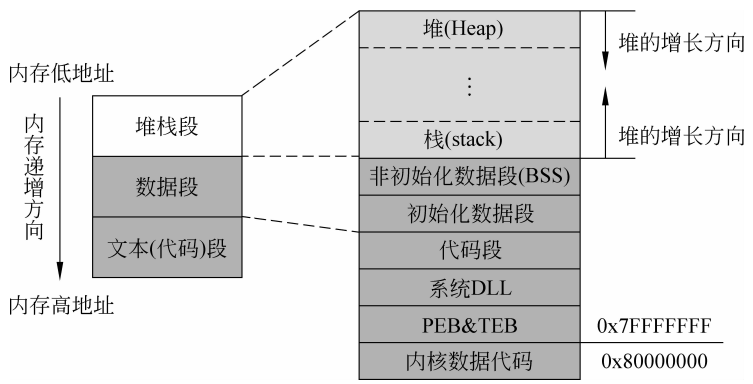

图 4-1 内存分布示意图

此处有3个重要的寄存器：堆栈指针寄存器(ESP)、基地址寄存器(EBP)和指令指针(EIP)。下面分别介绍这3个寄存器的作用。EIP(32位指令指针)是下一条指令在代码段(CS:指令或代码存放的位置)里的偏移，一般代码是不能直接访问EIP值的。在每条指令被执行之后EIP值会自动增加一个值以指向下一条要执行指令的地址。当要调用子程序时，系统就需要知道下一条指令的地址以及如何返回原始调用地址。调用指令通常规定了要往EIP所加的值，并把它压入堆栈。而调用函数中的返回指令会把堆栈值弹出给EIP以恢复调用后下一条指令的执行。同样堆栈指针ESP指向堆栈的最高端，EBP可指向堆栈的任意位置。换句话说，EBP是具有固定偏移的局部变量和参数堆栈信息的偏移参考寄存器；ESP地址经常发生变化，因为它一直指向堆栈的顶端。任何压栈和出栈操作均会影响ESP的值。

除此之外，必须理解函数的调用过程。通常一个函数调用过程有如下步骤：把参数压入栈，保存指令寄存器中的内容，作为返回地址，放入堆栈当前的基址寄存器，把当前的栈指针(ESP)复制到基址寄存器，作为新的基地址，为本地变量留出一定空间，把ESP减去适当的数值。

4.1.2 缓冲区溢出攻击原理

缓冲区溢出攻击发生的原因是因为缓冲区在栈中分配，然而复制的数据过长，因此覆盖了函数的返回地址或其他一些重要数据结构、函数指针，从而导致了溢出攻击的发生。

4.1.3 缓冲区溢出防御方法

避免缓冲区溢出攻击大致有以下几种方法：堆栈不可执行，寻址空间随机分布，对源代码进行安全漏洞分析，改善编译器，以及使用安全的编程语言和更安全的函数库。

1. 堆栈不可执行

由于现在大多数缓冲区溢出攻击基本上都是堆栈溢出攻击，其通过将攻击代码植入到堆栈中，然后在执行攻击代码。因此很容易想到一种最为简单的防御方法就是设置堆栈区不可执行。

2. 寻址空间随机分布

由于特定系统函数的入口地址是可以事先确定的。因此可以通过寻址空间随机分布来避免发生溢出攻击。在Windows在它的最新操作系统中采用了ASLR技术来防御此类缓冲区溢出，在操作系统启动时，随机从256个地址空间中选出一个载入DLL/EXE。这样攻击方就难以事先确定系统函数的入口地址。

3. 对源代码进行安全漏洞分析

利用专用安全分析工具软件，对源代码进行词法、语法上的分析，根据已知缓冲区溢出漏洞的数据库，找出程序中可能存在的安全隐患，并给予相应的解决提示，帮助编程人员迅速地修改程序。

4. 改善编译器

通过改进编译器，增加缓冲区完好检验、输入数据的边界检查等，防止缓冲区溢出，让

存在隐患的程序编译不能通过。这种方法为防止缓冲区溢出攻击提供了较好的解决方法，但是它需要程序的源代码，需要对其重新编译，增加了程序运行时的额外开销，存在一定的误报率，并且没有真正除去程序中的安全漏洞，当程序在普通环境运行时，漏洞依然存在。

5. 使用安全的编程语言

缓冲区溢出攻击发生的重要原因之一是使用了不安全的函数，例如 strcpy、scanf 等。程序员在调用这些函数时，如果没有进行仔细的检查，就很容易留下受到缓冲区溢出攻击的漏洞。因此使用更安全的函数库对标准函数库中有漏洞的函数进行封装，加入安全检查。安全工具以动态链接库的形式存在于系统中，对于用户是透明的。当程序调用有缺陷的函数时，它们会自动加载安全检查，通过检查后才真正去调用该函数。

4.2　实验　缓冲区溢出攻击实验

【实验目的】

通过对 OllyDbgICE 的使用，理解缓冲区溢出攻击发生的原理。

【实验内容】

根据实验步骤，一步步地实践和理解缓冲区溢出的原理。

【实验环境】

PC 一台。

【实验参考步骤】

(1) 编译如下简单程序例子，生成 DEBUG 版程序。

```
#include <stdio.h>
#include <windows.h>
#include <string.h>
int fun(char * cpybuf){
    char Buf[8];
    strcpy(Buf,cpybuf);
    return 0;
}
int main(){
    MessageBoxA(NULL,"This is a Test","BOF",MB_OK);
    char buff[]="1234567";
    fun(buff);
    return 0;
}
```

(2) 打开 OD，将上述生成的 DEBUG 程序 sample. exe 在 OD 中打开，图 4-2 是打开 sample. exe 的截图。

(3) 找到 main()函数的入口地址。对比代码，可以知道在地址 010F1488 处调用了

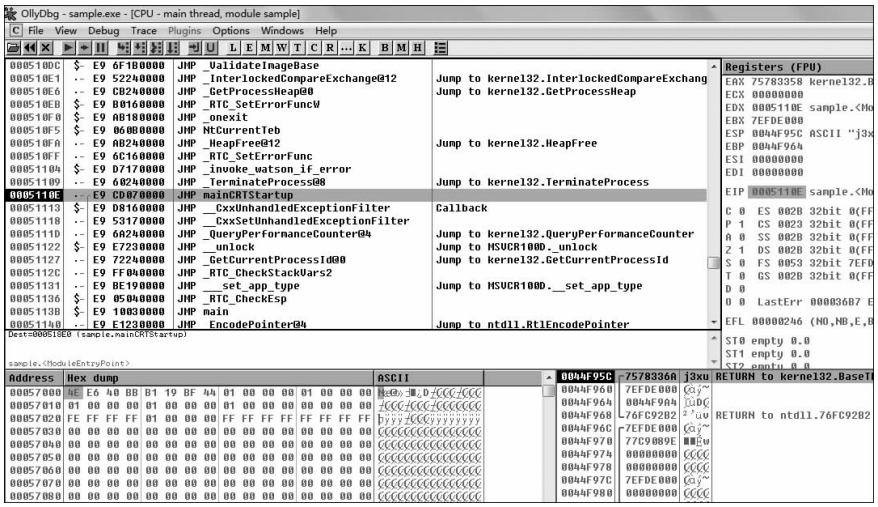

图 4-2　打开 sample. exe

动态链接库 User32. dll 中的 API 函数 MessageBox()，之后的连续的 MOV 指令则是初始化数组操作，并将初始化好的数组地址传递给 EAX，由它做参数传递，如图 4-3 所示。

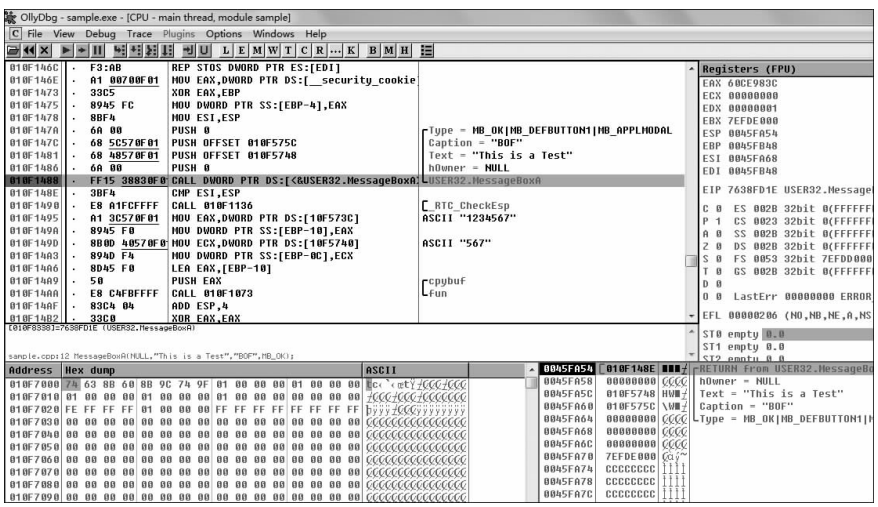

图 4-3　找到 main()函数的入口地址

(4) 单步跟踪至 010F13A0，即将进入 fun()函数，应注意堆栈的变化情况。F7 单步跟踪进入到 fun()函数内部，如图 4-4 所示。

显示了整个 fun()函数的反汇编代码情况。此刻注意堆栈的栈顶的情况，栈顶 0014F678 存放内容为 010F14AF，这正是图 4-3 中 main()函数中执行 fun()函数的下一句代码的地址，紧接着继续跟踪，看是否这个栈顶的内容是用于返回 main()函数。

(5) 单步跟踪到 fun()的最后一条指令 ret n 查看程序是如何回到 main()函数的，单步 F7 执行至 011714E2，指令为 RETN，查询汇编指令手册可知，是将栈顶值出栈给 EIP，如图 4-5 所示。

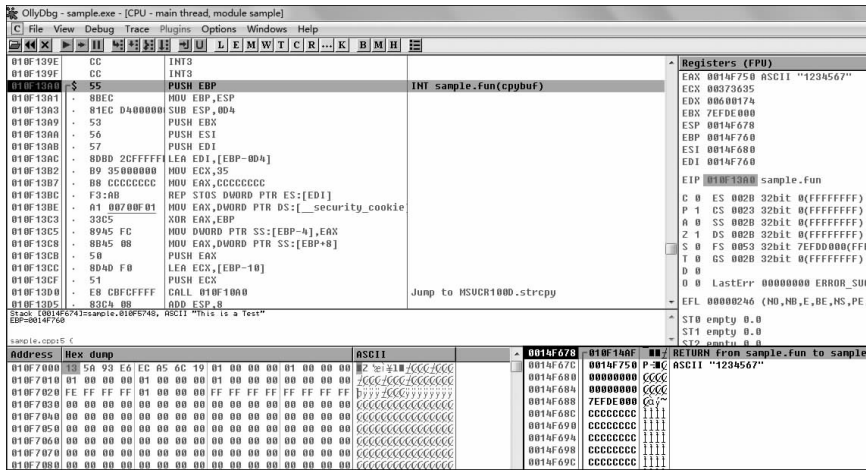

图 4-4　单步跟踪进入到 fun()函数内部

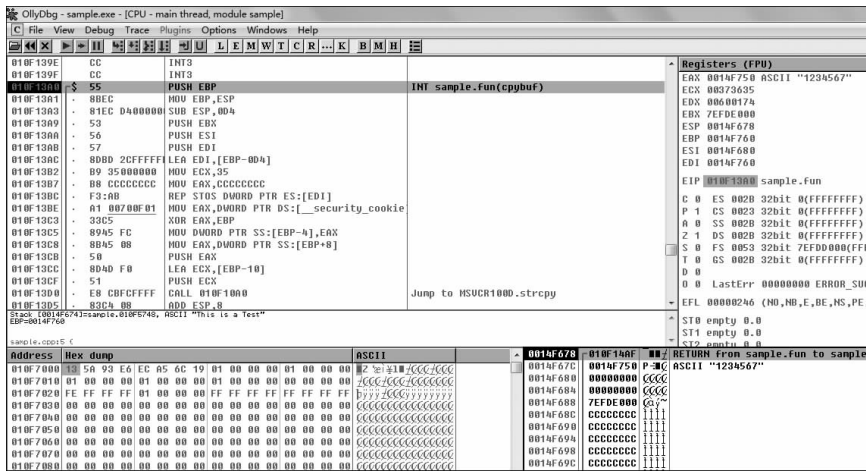

图 4-5　单步跟踪 ret n 指令

(6) 开始攻击演示。

修改源代码中的 buff[]初始化定义 char buff[]="1234567"为 char buff[]="1234567aaaaaaabbbbbbbbbbb",编译生成相应的 Debug 程序,然后放在 OD 里面根据上述步骤进行调试,注意栈顶的返回地址,如图 4-6 所示。

这是改变数组内容后的 fun()的反汇编代码,注意到栈顶 0013FF18 存放 004010D9,这是 main()中执行完 fun()函数后执行的代码。

(7) strcpy 执行。

单步执行到 Call 00401100,这里执行的是函数 strcpy(),查看执行 strcpy 后的结果,如图 4-7 所示。

此时可以发现 EBP 的值为 0014F7A8(ASCII 码为 bbbbbbbb)。

(8) 攻击发生。

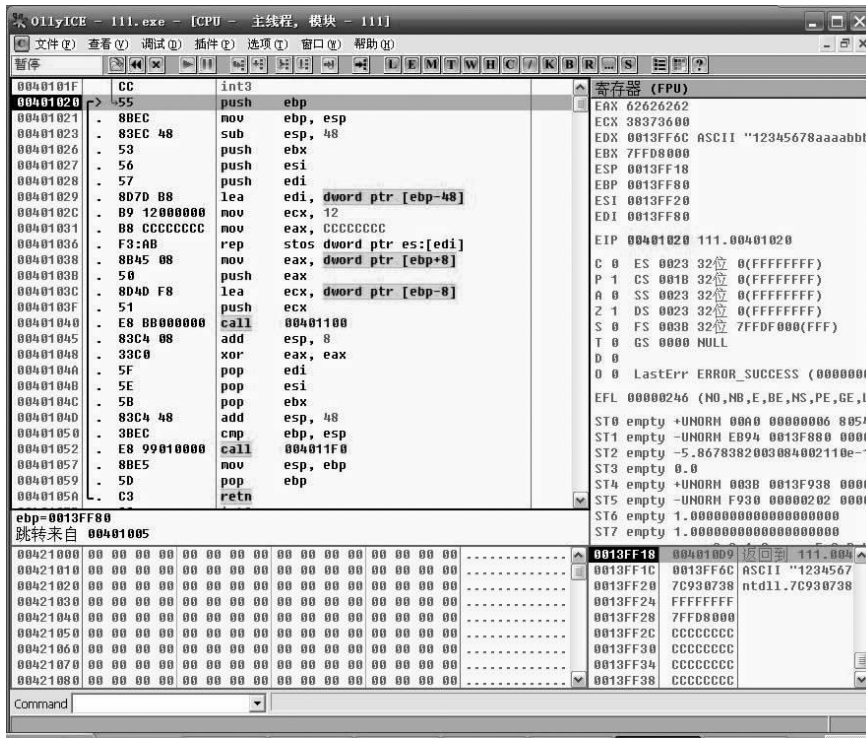

图 4-6　开始攻击

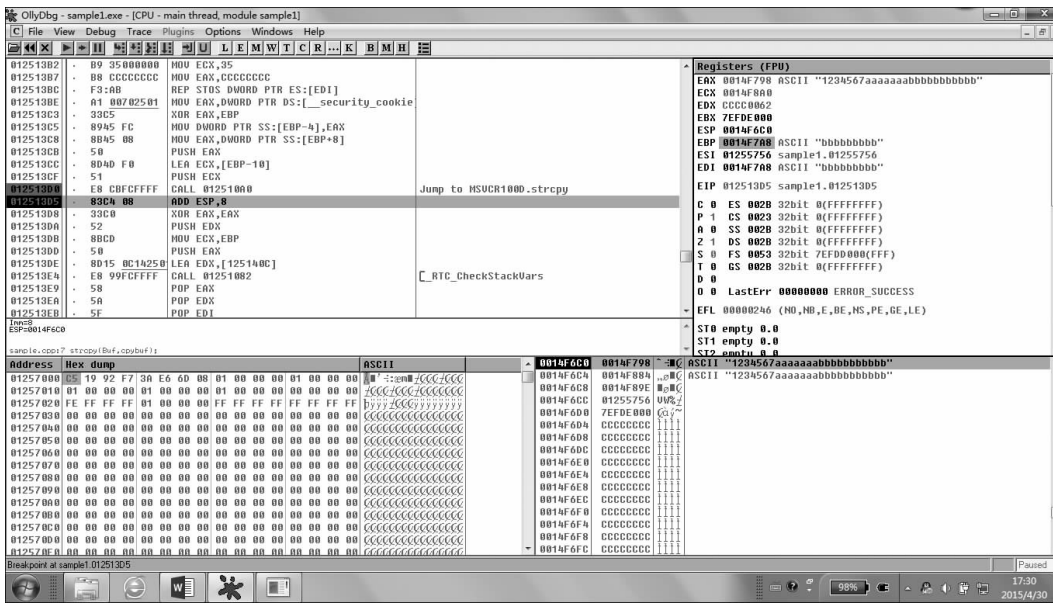

图 4-7　strcpy 执行

继续单步执行到 ret n 指令，栈顶正是 62626262，如果再执行 EIP->62626262，由于 62626262 不存在内容，导致程序执行出错，如图 4-8 所示，攻击发生。