摘　要

自从 1969 年互联网诞生以来，网络安全问题就一直是一个巨大的挑战。一方面，攻击者不断尝试寻找新的漏洞来入侵系统，另一方面，系统的开发者则试图通过各种手段来检测和阻止网络攻击来保护系统的安全。就像矛和盾一样，二者之间的战争从未停止。

如今，随着互联网特别是移动互联网和物联网的飞速发展，网络安全的形势正变得越来越严峻。而网络逃逸技术作为一种常见的攻击手段，常常和病毒木马结合，逃过 IDS(入侵检测系统)的检测，严重危害着我们每个人的安全。

本文从一个攻击者的角度，介绍了逃逸技术和 AET(高级逃逸技术)相关的技术，包括常见的逃逸方法，逃逸工具，以及逃逸技术的防范措施。并且实现了一个 TCP/IP 层的高级逃逸系统，用来线下发现 IDS 的弱点，帮助改善 IDS 的性能。

该系统以插件的形式运行在中科院信工所的大规模实时流量处理平台 SAPP 上，系统一共实现了 8 种网络逃逸的方法，并且支持用户自定义逃逸方法的组合。系统采用模块化方法编写，每种逃逸方法都是一个模块，可拓展行很强。

关键词：IDS　逃逸技术　SAPP　AET

Abstract

Since the birth of the Internet in 1969, network security has been a great challenge. On the one hand, attackers continue to try to find new vulnerabilities to invade the system. On the other hand, system developers try to protect the security of the system through various means to detect and prevent network attacks. Like spears and shields, the war between the two never ends.

Nowadays, with the rapid development of Internet, especially mobile Internet and Internet of things, the situation of network security is becoming more and more serious. As a common means of attack, network evasion technology often combines with virus trojan, and escapes the detection of IDS (Intrusion Detection System), which seriously endangers the safety of each of us.

From an attacker's point of view, this paper introduces the related technologies of network evasion technology and advanced evasion technology, including common evasion methods, evasion tools, and prevention measures of evasion technology.

And we implement a advanced evasion technology system on TCP/IP layer, which is used to detect weaknesses of IDS,and to help to improve the performance of IDS.

The system runs as a plug-in in the Chinese Academy of Sciences Institute of large-scale real-time traffic signal processing platform called SAPP. The system achieves a total of 8 methods of network evasion, and supports for the combination of user-defined escape method. The system is written in a modular way, and each escape method is a module, which can be expanded very easiy.

Keywords: IDS Evasion Technology SAPP AET

目　录

第 1 章　引　言

1.1 网络安全现状

自从 1969 年互联网诞生以来，网络安全问题就一直是一个巨大的挑战。一方面，攻击者不断尝试寻找新的漏洞来入侵系统，另一方面，系统的开发者则试图通过各种手段来检测和阻止网络攻击来保护系统的安全。就像矛和盾一样，二者之间的战争从未停止。

如今，互联网，特别是移动互联网和物联网正在飞速发展。据统计，截止到 2016 年 12 月份，我国网民规模已经达到 13 亿人，2016 年全年共计新增加网民人数 4299 万。互联网普及率已达 53.2%，相比较 2015 年底提升了 2.9 个百分点。预计 2017 年底我国网民规模将要达到 7.72 亿，互联网普及率将达到 55.9%。

图 1-1　中国网民规模及互联网普及率情况

另一方面，随着互联网的飞速发展，网络安全的形势正愈发严峻。2016 年 360 互联网安全中心发布了《2016 年中国互联网安全报告》（以下简称《报告》）。《报告》中对个人与政企所面临的电信骚扰、恶意程序、网络诈骗、钓鱼邮件、网站安全、IoT 安全、DDOS 攻击、安卓系统漏洞、工控

安全、邮件安全、网络扫描应急响应、APT 等等安全威胁情况做了全面的介绍。报告显示，在对超过 197.9 万网站的漏洞检查中发现，国内大概有 43.6%的网站存在着安全漏洞，并且在这其中高危漏洞占到 7.1%。网络安全形势不容乐观。

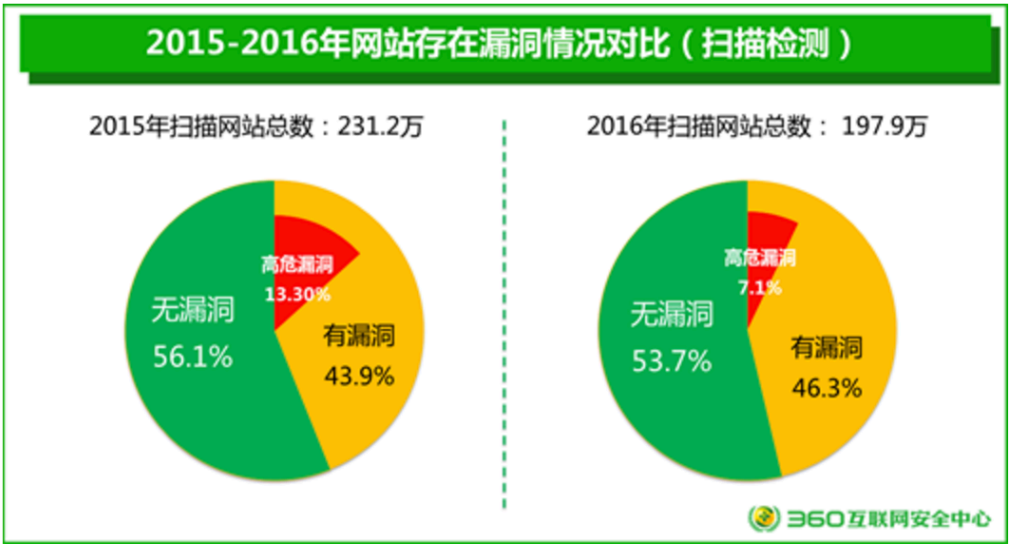


图 1-2　2015-2016 年网站存在漏洞情况对比

同时，用户的网络安全意识非常薄弱。据不完全统计，我国有 81%的网民不定期更换账号密码，而在这其中，真正遇到问题才去更换密码的达到了 64%，从来不换密码的达到了 17%。另外，有 44%的网民使用姓名、生日全拼或者电话号码作为密码，而青少年占比更高，达到了 49%。另外，75%的网民存在着多个账户使用同一个密码的问题，而在这其中，青少年的问题更为严重，占比达 82%。

同时，用户扫描二维码太容易中招，据不完全统计，36.96%的网民对二维码的回答是“经常扫，但是不考虑是否安全”，在这其中，青少年网民中对二维码“经常扫，但是不考虑是否安全”的比例高达 40.3%。

另外，有超过 80%的用户随意连接公共 Wifi，因此用户支付行为存在这很大的风险，据不完全统计，有 80.2%的网民不经安全检查就随意连接公共 Wifi，在这其中，45%的网民不但使用免费公共 Wifi 浏览网页，而且使用即

时通信工具如微信，短信等。如今网上支付越来越普遍，但是据统计有 83%的网民的网上支付的行为存在着严重的安全隐患。在这其中，有 38%的网民使用没有密码的公共 Wifi 进行网络支付，有 42%的网民在公共网络进行支付后没有选择消除上网痕迹。

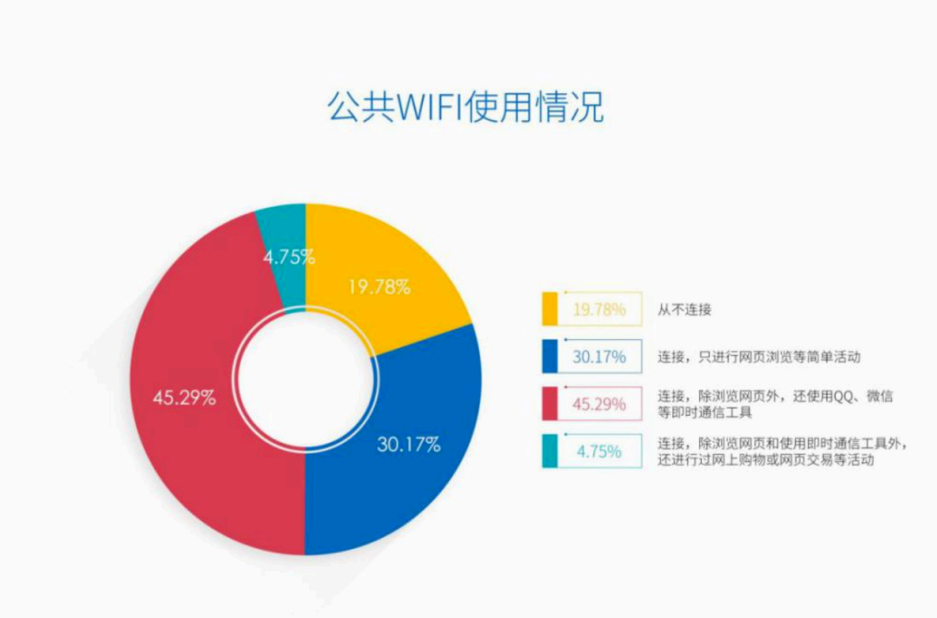


图 1-3　公共 WIFI 使用情况

2013 年爆发了一件轰动全球的”斯诺登事件“和“棱镜门”，让人们更加地刻地意识到网络安全的重要性。正因为如此，2014 年 2 月 27 日，中央成立了网络安全和信息化领导小组，由国家主席习近平亲自担任组长。该领导小组将着眼于国家安全，统筹协调包括政治、经济、文化、军事等各个领域的信息化和网络安全等重大的问题，研究制定信息化和网络安全的发展战略和重大政策，从而推动国家信息化和网络安全的建设，不断增强我国的安全保障水平。习主席更是提出“没有网络安全，就没有国家安全”的战略，将网络安全提升到国家安全的高度。

1.2 常见网络攻击方法

网络攻击方法多种多样，层出不穷。本章只介绍几种常见的网络攻击方法及其对应的防范措施：SQL 注入，DOS 攻击，恶意代码攻击，钓鱼网站等。

1.2.1 SQL 注入攻击

SQL 注入攻击是黑客在对数据库进行攻击时使用的常用手段之一。因为程序员的水平参差不齐，部分程序员在写代码的时候，并没有对用户输入数据是否合法进行判断，导致了应用程序存在安全隐患。用户可以提交一段看似正常的数据库查询代码，通过程序返回的结果，获得某些期望的数据，这就是人们常说的 SQL Injection，即 SQL 注入。

由于 SQL 注入利用的是正常的 HTTP 请求，所以表面上看起来和正常的 HTTP 请求没有任何区别，隐蔽性极强，很难被发现。

SQL 注入一般分为下面几个步骤：

第一步：判断当前的环境是否可以进行 SQL 注入。如果仅仅是访问网页，不涉及到数据库操作，那么就不存在 SQL 注入的问题。

第二步：寻找 SQL 注入点。第一步通过后，就可以去寻找 SQL 注入点了。一般是先通过构造特殊的输入，然后根据浏览器返回的内容猜测数据类型，然后就可以想办法找到注入点。

第三步：猜测数据库用户名和密码，攻击者可以在网上找到很多现成的注入工具来破解用户名和密码。

第四步：获取网站后台管理入口，一般这个入口不对用户开放，攻击者可以利用扫描工具扫到可能的入口然后依次去尝试登录直到成功为止。

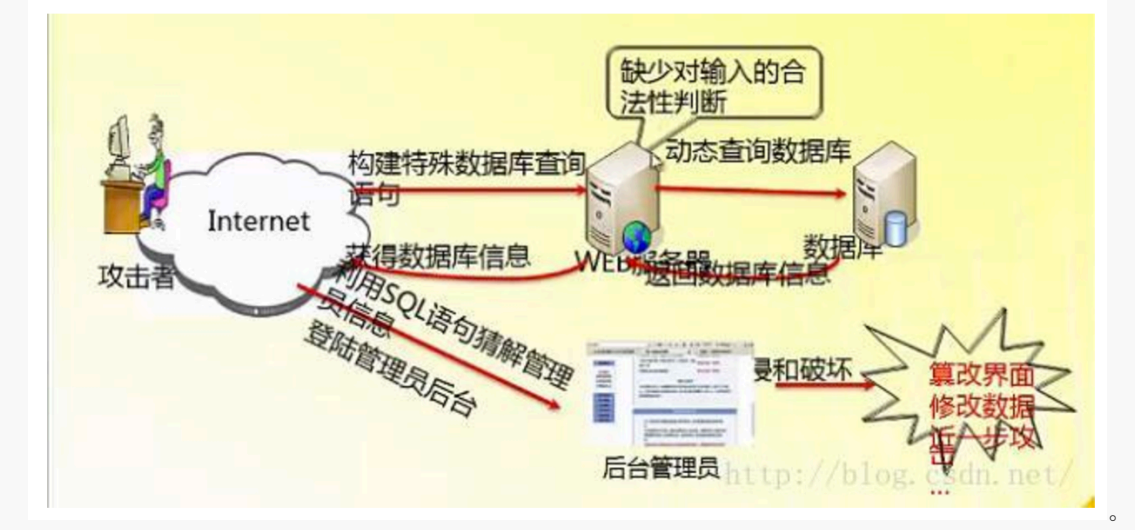


图 1-4　SQL 注入一般流程

1.2.2 DOS 攻击

DOS 攻击（Deny of service）又叫拒绝服务攻击。通过耗尽目标系统资源使目标系统无法提供正常的服务甚至瘫痪。被攻击的系统资源包括 CPU，内存，网络带宽等。这是一种很常见的攻击，全球平均每小时遭受 28 次 DOS 攻击。DOS 攻击的主要表现方式以下 3 种：

第一种：构造大量无用连接，导致通往目标主机的网络带宽被打满，是正常请求无法与目标主机进行通信。

第二种：利用目标主机或者传输协议的缺陷，高频发送重复请求，是目标主机无法及时处理正常的请求。

第三种：利用目标主机或者传输协议的缺陷，反复发送异常的攻击数据，使目标主机错误分配大量的系统的资源，从而使主机瘫痪。

常见的 DOS 攻击方法包括 Synflood，Smurf，Land_based，Ping of Death，Teardrop，PingSweep，Pingflood。具体的每种方法不在本文讨论范围之内。 在 DOS 的基础上，还产生了 DDOS 攻击（Distributed DOS 攻击），即分布式拒绝服务攻击。它指的是在拒绝服务攻击的基础上，通过分布式网络，操纵大量计算机同时向目标主机发起攻击。通常 DDOS 攻击的计算机数量都可以达到百万甚至千万台，从而极大地提高了攻击的效率和成功率。

值得一提的是，攻击者一般通过木马病毒感染获得含有大量计算机的僵尸网络，然后操纵这些僵尸网络同一时间发起 DDOS 攻击。具有很大的威胁性。

1.2.3 恶意代码攻击

恶意代码攻击指的是能够在计算机中进行非授权操作的代码，包括各种木马，病毒，缓冲区溢出攻击。恶意代码攻击威胁最大，种类最多，传播性强，难以防范。以木马病毒举例，木马病毒分为控制端和被控制端，用户一旦感染木马病毒，攻击者就能享有目标主机的大部分操作权限，包括修改文件，修改注册表，控制鼠标，键盘等。同时由于木马病毒具有很强的隐蔽性，攻击者通常会采用各种方法来伪装木马病毒，使其看起来和正常的软件一样，所以即使用户发现感染木马病毒也很难及时找到并删除。

恶意代码攻击的危害很大，可能会盗取受害者电脑中的重要信息，包括网银账号，密码。可能会加密整个磁盘，从而进行敲诈勒索。而且，恶意代码层出不穷，而且有很多变种，对于一个新型的病毒，基本上没有办法防范。基于此，恶意代码攻击已经成为网络安全的最大威胁。

1.2.4 钓鱼网站

钓鱼网站通常指的是伪装成银行及电子商务网站的“假冒”网站，通常用来窃取用户提交的银行卡，账号，密码等信息。钓鱼网站一般通过电子邮件进行传播，在邮件中一般有一个经过伪装的链接来诱惑用户去点击。钓鱼网站的页面通常做的和真实网站界面完全一样，只是 URL 有细微差别，要求用户提交账号密码等信息。一般来说，钓鱼网站的技术含量比较低，攻击者只需要制作几个界面和一个和真实网站 URI 非常相似的 URL 即可。但是用户如果不注意甄别，就很容易上当受骗。

1.3 入侵检测系统

IDS（Intrusion Detection System）又叫入侵检测系统。它指的是按照一定的安全策略，对系统的运行状况进行监控，尽可能及时发现各种攻击行为，然后进行报警。从而使用户可以提前发现各种安全风险，然后及时进行处

理。IDS 的存在有效地保证了系统的安全性，极大地降低了系统被恶意攻击的风险。在一些大型的系统或者网络中，IDS 是一种必须的配置。

IDS 与传统的防火墙不同，IDS 不进行拦截，只是进行监控，报警。作为一个旁路监听设备，IDS 不需要部署在必须的网络链路中，只需要从需要监控的网络流量中通过“分光”的方式拷贝一份给 IDS 即可，不会影响真实的流量，这被称为 IDS 的“并联”模式。

图 1-5 简单的 IDS 示意图

一个 IDS 是否高效，主要取决于 IDS 中所采用的入侵检测方法，目前实际应用中所采取的入侵检测方法主要由以下几种：

1）模式匹配，英文称为“signatured-based”，即基于特征的入侵检测方法。这种方法通常情况下会建立一个攻击特征库，里面存的都是目前已知的攻击的一些特征（比如特定的代码段，特定的命令等），然后针对流经的网络流量，依次检查发送过来的数据报文是否包含这些特征。这是最传统的入侵检测方法，与其他方法相比，它简单而且高效。但是缺点也很明显，它只能检测已知攻击，并且检测效果完全依赖于特征数据库，有很强的局限性，同时，由于要分析处理大量的网络流量，这种方法的效率将成为瓶颈。

2）统计模型，统计模型通常用于异常检测，在统计模型中通常采用以下 3 个测量参数：间隔时间，审计事件数量，资源消耗情况。常用的 5 种统计

模型为：a、操作模型，该模型将统计结果和标准指标相比较来发现异常，标准结果通常通过统计一定时间内的平均值或者根据经验得到。举个例子，如果在短时间内出现多次失败的登录口令，那么可能是攻击者在尝试枚举口令进行攻击。b、方差，先设定好置信区间，然后计算测量结果的方差，当方差超过置信区间的范围时，那我们就可以怀疑这是个异常 Case。c、马尔科夫过程模型，我们可以将每种类型的事件都定义成为系统状态，然后通过状态转移矩阵来表示状态的变化。当一个事件发生时，我们可以查看状态转移矩阵中该转移的概率，如果概率较小，那么我们可以怀疑这是一个异常 Case。d、时间序列分析，我们可以将事件计数根据时间排成序列，如果某个时间在某个时间发生的概率比较低，那么我们可以怀疑这是一个异常 Case。

3）文件完整性检查，这种入侵检测方法通常是将系统目前的文件和上次检查时的文件进行比较，如果发现了有文件变化，那么可以怀疑系统受到了入侵。通常来说，文件完整性检查系统中保存着每次检查的结果。每次检查时，将待检查的文件进行 Hash 得到一个文件摘要，然后将文件和摘要对应关系存在数据库中，方便下次进行对比。

4) 基于机器学习的入侵检测方法，随着机器学习的飞速发展，各个领域都在利用机器学习来提升效果。入侵检测方面也不例外，基于机器学习的入侵检测有分为有监督和无监督两种，有监督的机器学习将每个数据打好标签做成数据集，用于机器学习算法的训练。无监督的机器学习只需要将所有数据作为输入扔给机器，让它自己去寻找规律。基于机器学习的入侵检测方法不需要对网络安全有很深的研究，也不用花费大量时间去寻找攻击特征，只要有数据即可。随着机器资源和机器学习算法的普及，这种方法未来将变得越来越流行。

1.4 逃逸技术

逃逸技术，英文被称为“Evasion Technology”，是一种伪装网络攻击的手段，它是指攻击者通过运用各种手段对数据包进行处理，来使攻击代码能够逃过 IDS 的检测。从而达到危害目标主机的目的。

逃逸技术最早出现在 1998 年。1998 年 1 月 Ptacek 和 Newsham 发表了名为“Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”的论文，描述了 IDS 产品和模型存在一些基本面上的问题及如何从 TCP/IP 底层（网络层）绕过 IDS 检测的一系列方法。其主要思想是利用 IDS 对数据报文的分析处理方式与目标主机处理方式的不同，进行插入、逃避及拒绝服务攻击，从而使 IDS 无法正确地检测到攻击。这是网络安全届第一次提出逃逸技术这个概念，也正因为此，这篇论文已经成为逃逸技术相关的经典论文。

1988 年发表的这篇论文重点在于 IP 碎片带来的挑战，里面也提到了其他逃逸技术，包括修改 IP 选项，修改 TCP 选项等。知道今天，该论文中所提到的许多逃逸技术仍然可以有效避开现在的 IDS 系统的检测。

1.5 高级逃逸技术

高级逃逸技术，英文叫做 AET（Advanced Evasion Technology）。是在逃逸技术的基础上演变出来的一种逃逸技术。

2010 年 8 月，芬兰的 StoneSoft 公司宣布发现了一种新型的高级逃逸技术（AET）。AET 的发现极大拓展了现有的逃逸技术知识，随后，StoneSoft 将此次重大发现详细报告给芬兰的 CERT（计算机应急处理中心）。并且，ICSA（国际计算机安全协会）对其进行了证实，研究发现，现有的许多网络安全解决方案都无法检测这种高级逃逸方式。

和传统逃逸方法相比，AET 通常具有以下特点。a、基本原理没有改变 b、以普通逃逸为基础，是多个层次多个方法的组合 c、拥有更强的逃逸能力，很难被 IDS 检测 d、逃逸方法数量巨大

1.6 本文的安排

本文第一章先是介绍了网络安全现状，常见的网络攻击手段，以及入侵检测系统的相关知识。帮助读者快速了解目前网络安全领域面临的威胁和常见的网络攻防手段。然后简单介绍了本文要论述的重点，即高级逃逸技术。

第二章详细介绍了各种逃逸技术的分类及其应用场景。除此之外，还介绍了能够实现这些逃逸技术的软件。

第三章介绍了系统设计工作，主要包括控制器设计，与 SAPP 的交互设计和具体的逃逸方法的设计。

第四章介绍了平台的实现部分，包括控制器的实现，以及各种逃逸方法的实现。

第五章介绍了实验部分，包括实验步骤，实验结果，实验结果的分析等。

第六章总结了本文的主要工作并对下一步工作做出了展望。

第 2 章　逃逸技术介绍

2.1 逃逸技术分类

2.1.1 拒绝服务

首先是拒绝服务攻击：目的是淹没网络带宽和耗尽系统资源比如 IDS 的 CPU 和内存空间。除了生成大量的网络流量，攻击还可以利用检测算法的弱点进行攻击。在这种攻击中，攻击者通过回溯法去尝试覆盖所有可能的规则匹配来让算法每次都计算最坏的情况。在这种情况下，实验证明：只要 4.0kbps 带宽的流量就可以进行一次 DoS 攻击。这个利用算法复杂度进行攻击而不是淹没网络带宽是 DoS 攻击的一个典型例子。

2.1.2 包分片

包分片包括 IP 包分片和 TCP 包分段，它把 IP 数据包或者 TCP 流分成非重叠的分片或者分段。如果 IDS 不能完全重组包的内容，就可能会忽略一个针对目的主机的攻击。举个例子，IPS 可能会在包负载中寻找特征 /bin/sh，但是攻击者可能会将包分成两片，一片包括 /bin，一片包括 /sh。如果 IPS

不能重组这两个段，那么它就不能发现发现这个特征，从而使攻击者逃避检测。

因为 IDS 负责监控所有的网络流量，理论上来说，它必须重组网络中所有的 IP 分片和 TCP 分段来防止逃逸。但是 IDS 对每个连接的信息有资源限制。举个例子，用来重组的缓冲区可能是不够的。因此，IP 分片或者 TCP 分段可能是有效果的，如果 IDS 恰好不重组的话。

2.1.3 重复包插入

重复包插入是指插入重复的包或者重叠的 TCP 分段或者 IP 分片用来迷惑 IDS。这个技术依赖于 IDS 和目的主机对重复/重叠的分片的处理方式不同，因为 IDS 缺乏相关的信息比如网络拓扑和目的主机的操作系统。图 1 用一个具体的例子说明了这个技术如何工作。在这个例子中，攻击者插入具有小 TTL 值的段，所以它在到达目的主机前会被丢弃。

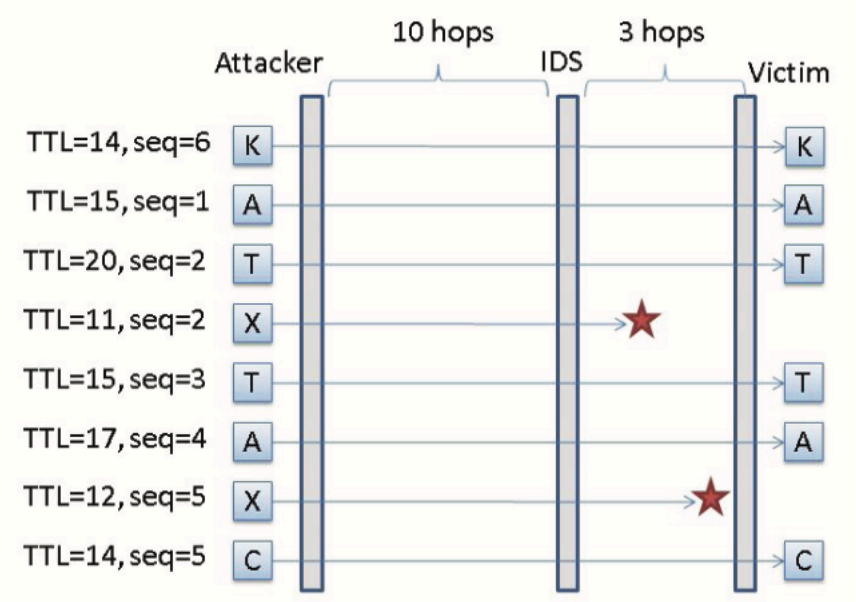


图 2-1　重复包插入示意图

重叠的 IP 分片和 TCP 分段可能对 IDS 是模棱两可的。举个例子，假设一个 TCP 段的序列号是 10，负载 ATTXYZ，另一个段的序列号是 13，负载 ACK。当收到这两个包时，目的主机可能会重组成 ATTXYZ 或者

ATTACK，这取决于主机的操作系统。如果不知道主机的操作系统，那么IDS 处理这两个包的方式可能会和主机不一致。攻击者就可以利用这个来逃逸。

2.1.4 负载变换

负载变换意味着攻击者将恶意的负载转换成语义上等价的负载。被转换的负载看起来和 IDS 期待的特征不同，所以攻击者可以逃逸检测。因为被转化的负载在语义上是相同的，所以攻击仍然是有效的。举个例子，利用 libwhisker（www.wiretrip.net/rfp/txt/whiskerids.html）这个库，一个 HTTP 请求的 URL 头可以被转化成等价的表达。这个转换可以是 16 进制编码。用 URL 的 16 进制编码做例子，如果一个目录名 cgi-bin 被编码成 %63%67%69%2d%62%69%6e，IDS 检测 cgi-bin 就会失败。一个 SQL 注入攻击可以用同样的方法逃逸检测。

和重复包插入一样，负载变化对 IDS 来说也是模棱两可的，因为主机和 IPS 的处理方式不同。举个例子，阿帕奇服务器不能接受反斜线作为合法的斜线，但是微软 IIS 服务器可以。因此，管理员不得不配置 IDS 来使其和主机的处理方式一致。

2.1.5 Shellcode 变换

Shellcode 变换将 Shellcode 编码成多态的形式来逃逸 IDS 的检测。有许多方法来实现这一点。举个例子，攻击者可以加密或压缩 Shellcode，并且预置解密或解压机在代码中。攻击者还可以用和原始代码语义相同的代码来取代原代码，一个具体的例子是插入 nop 指令来让代码看起来不同。比如一个指令 mov eax, ebx 可以用 push ebx, pop eax 来代替。这样基于特征的 IPS 将检测不出来，从而实现逃逸。这个技术也被用在各种恶意软件中比如病毒和蠕虫。

因为有太多方法来变化一个 Shellcode，对 IDS 来说，检测多态的代码变得特别困难。一个 IDS 可能需要加密解密代码甚至模拟代码运行（比如在沙盒运行）来发现恶意行为。因此还原 Shellcode 就变得特别耗费计算资源，甚至达到 IDS 的负载。

2.2 逃逸工具

逃逸方法多种多样，逃逸工具也层出不穷。本文总结了几种常见的逃逸工具以及每种工具各个的适用场景。

Fragroute，在TCP/IP层实现了包分片和重复报插入。攻击者在启动Fragroute之前只需简单写个脚本配置一下需要用到逃逸技术的顺序，然后系统就会按照配置依次执行这些逃逸技术。

Nikto，一个可以生成很多有害URI请求的网络扫描工具。它可以帮助开发者和网络管理员测试服务器上可能的安全问题。这个软件也提供一个反IPS的方法：负载变化来使网络扫描器逃避IPS的检测。

ADMmutate，是一个实现shellcode突变的工具，一个使用ADMmutate API的程序可以将shellcode转换成不同的形式来逃避基于特征的IPS的检测。

Sploit，是一个允许攻击者开发新的逃逸技术的逃逸测试框架。这个框架包括大部分Fragroute，Nikto，ADMmutate提供的逃逸技术。它也提供了应用层（如FTP，IMAP）的负载变换。举个例子，\xff\xf1是一个无用的telnet控制指令，所以一个FTP命令PASS就可以用P\xff\xf1ASS代替，这样就逃逸了匹配特征PASS的IPS的检测。

Metasploit，是一个渗透测试，IDS特征匹配，漏洞研究的框架。它支持利用漏洞对远程主机进行攻击。Metasploit框架还提供比ADMmutate和Sploit更多的shellcode变化[11]。

Havij，是一个利用网页漏洞的自动SQL注入工具。并且将攻击字符串编码成16进制，BASE64等。

逃逸方法	逃逸工具
包分片	Frogroute，Sploit
重复包插入	Frogroute，Sploit
负载变换	Nikto，Sploit，Havij
Shellcode变换	ADMmutate，Sploit，MetaSploit

图 2-2　逃逸工具适用范围

2.3 逃逸技术的防范

2.3.1 DoS 攻击的防范

通常来说，管理员可以增加可用带宽，使用 IDS 集群等来增强面对 DoS 攻击的健壮性。当 Dos 攻击发生的时候，管理员可以屏蔽特定的恶意流量并且调查这个攻击向量。然后向 IDS 开发商报告，来检查 IDS 对于算法复杂度攻击是否是脆弱的。

2.3.2 包分片和重复包插入的防范

应对这个攻击主要是靠 IDS 对包进行重组来还原原始包。现在的很多研究提出了特征匹配算法而不是重组包。这个算法的主要思想是将原本的特征也分成更小的特征，然后用这些更小的特征去匹配而不是重组包。但是这个算法并不解决其他逃逸方法比如负载变换。

目前有两个主要的技术来应对重复包插入。第一个就是流修改，通过在包到达 IDS 之前就选择一个方法来解释重复包，这样 IDS 和目标主机看到的就是同一个包，从而消除二义性。但是这种做法可能存在的问题是标准化过程中内存开销太大，因为需要将未回复的包都存在内存中。

2.3.3 负载变化的防范

要应对负载变换，IDS 必须要能够将转换后的负载还原出来。举个例子，当在 URI 请求中发现 16 进制编码 %20，IPS 必须能够将其还原成空格。一个 IDS 必须要能够处理不同应用中的编码方法。因为目标主机支持的编码方法各不相同，所以 IDS 必须仔细配置来确保 IDS 和目标主机的处理方法一致。另一个防护方法是基于主机的文本扫描，IDS 必须和后端的 web 服务器合作，当收到一个请求后，web 服务器先解码这个 URL，然后将结果返回 IDS 去检测。

2.3.4 Shellcode 变化的防范

有许多种方法来检测多态 shellcode，比如可以使用返回地址的范围来检测针对缓冲区溢出攻击的 shellcode。但是因为这个方法需要手动配置每个漏

洞的返回地址，所以并不可用，即使它的准确度非常高，误检率 0.01%，漏检率 0%。

还有一个基于神经网络的混合工具可以将分解的指令分类。它以指令的执行频率作为特性进行训练和分类。尽管这个方法的效果很好，但是它不能检测一个从未出现过的 shellcode。

网络层的多态 shellcode 检测在一个模拟器中（如沙盒）模拟 shellcode 的执行。这个方法将输入流当成指令并且试图执行。这个检测是可行的因为通常执行一个随机的字节会很快停止（遇到非法指令），但是多态 shellcode 只有当负载完全被解密时才会被执行。这个检测按照以下的策略：如果和位置有关的指令加密负载被找到而且在一小块内存中有大量的读写，那么可能是解密进程子在工作。

第 3 章　高级逃逸系统设计

系统目标是实现一个基于网络中间件的高级逃逸系统，要求能够实现 TCP/IP 层的大部分逃逸操作，并且支持各种逃逸方法的组合。在这里网络中间件指的是中科院信工所的 SAPP 平台。

基于此，本文实现了一个高级逃逸系统，该系统以插件的形式运行在 SAPP 平台上，平台运行时会自动加载该插件。插件运行时，通过 SAPP 提供的接口 API 获取网络流量，经过逃逸系统处理后，再通过 SAPP 提供的接口 API 发送出去。在整个过程中，逃逸系统只需要做最核心的逃逸处理即可，其他的操作比如 Pcap 捕包，发送数据包等操作都由 SAPP 平台完成，从而实现收发和处理操作解耦。

3.1 SAPP 平台简介

SAPP（Stream Analyse Process Platform）平台，是面向高速网络流处理的网络安全开发平台。具备全栈的协议解析能力，支持 IPV4/IPV6 网络层协议，支持 VLAN、GRE、MPLS、PPPOE 等中间层协议，支持 Socks、HTTP、SMTP、IMAP、POP3、FTP、Telnet 等应用层协议。在相应的硬件平

台上可以实现 10Gbps 到 40Gbps 的处理能力。支持并接和串接两种部署方式。

3.2 逃逸系统设计

逃逸系统主要分为 4 个部分：控制器，逃逸方法，SAPP 交互，配置文件。控制器是整个程序的主体部分，它负责从输入原始数据到输出处理后的数据的整个流程。逃逸方法指的是具体的方法，属于功能函数，每个逃逸方法都是一个单独的模块。SAPP 交互指的是调用 SAPP 平台的收发接口的部分，属于程序 IO 部分。配置文件指的是用户自定义逃逸方法及其组合的配置文件。

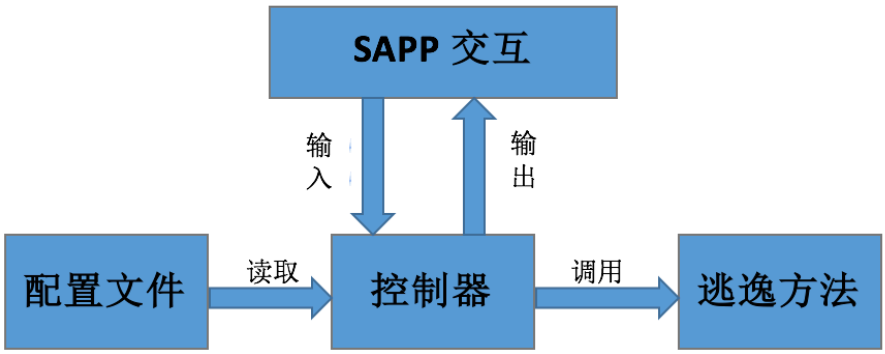


图 3-1 逃逸系统设计图

3.2.1 控制器

控制器是程序的主体部分，从收到一个数据包到经过处理后发送出去都由它负责。由于需要实现多种逃逸方法的组合，所以设计了 1 个队列来存储数据包，初始时队列里面都是待处理的包，每当处理一个包后，就将处理后的得到的新数据包加入队列，同时将原数据包扔出队列。等到逃逸命令全部完成后，队列中剩下的数据包就是经过逃逸处理后的数据包。对一个输入包处理的流程为：

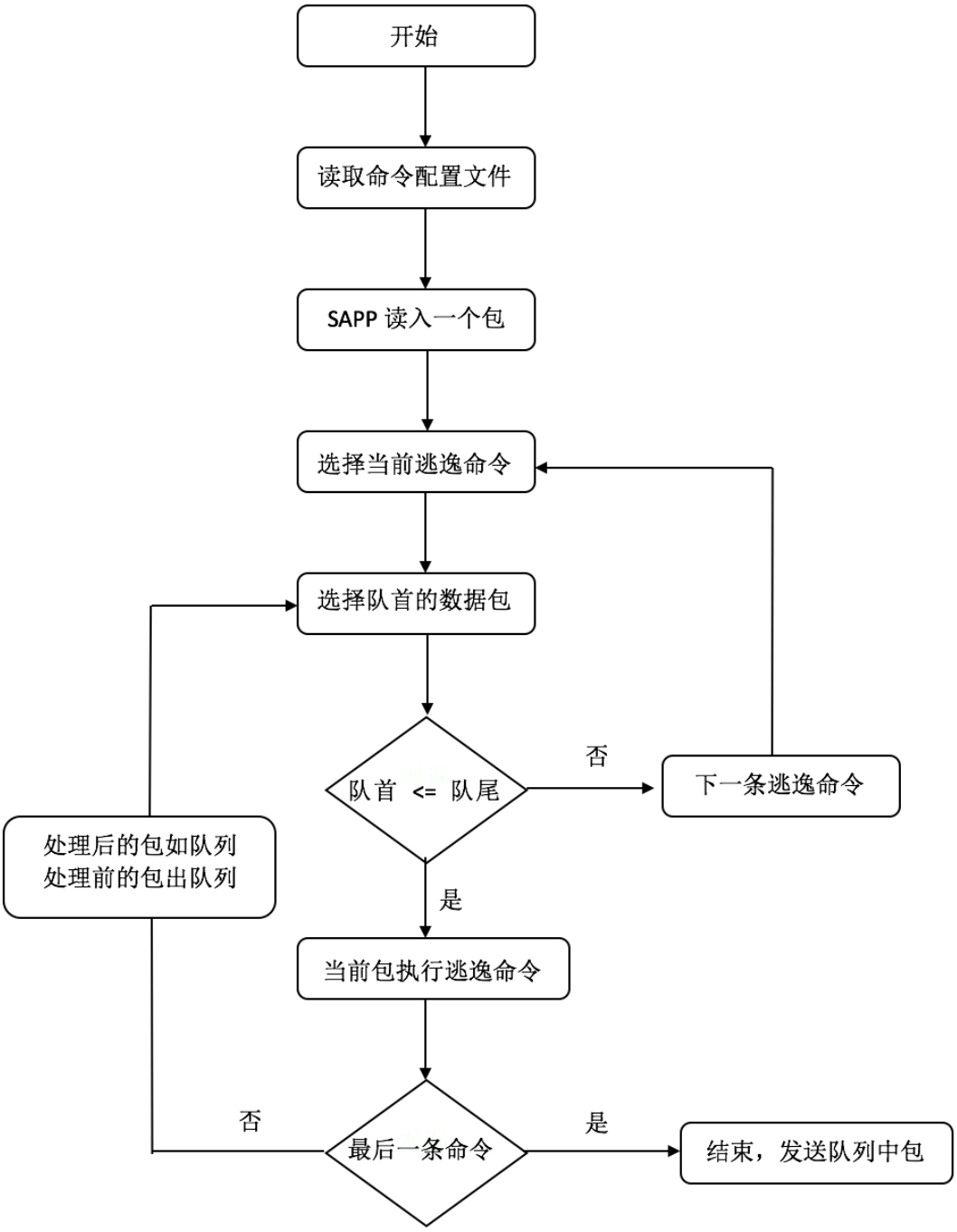


图 3-2 控制器处理流程

3.2.2 逃逸方法

本系统一共实现了 8 种 TCP/IP 逃逸方法，分别为 IP 分片，IP 重叠，TCP 分段，TCP 段重叠，修改 TTL，修改 MSS，修改 wscale，修改 TCP 的标志位。具体见下面的表格。

逃逸名称	对应模块名称
IP 分片	module_ip_frag
IP 重叠	module_ip_overlap
TCP 分段	module_tcp_frag
TCP 段重叠	module_tcp_overlap
修改 TTL	module_modify_ttl
修改 TCP 选项	module_modify_tcp_flags
修改 MSS	module_modify_mss
修改 wscale	module_modify_wscale

表 3-3 系统实现的 8 种逃逸方法

3.2.3 配置文件

由于是高级逃逸系统，所以系统支持不同逃逸方法的组合，用户可以自己在配置文件中配置需要执行的逃逸方法，执行顺序，逃逸参数等。配置文件需要放在当前目录下，命名为 AET.conf。

配置文件的命令配置规则类似于 linux 下的命令，即命令+参数。具体的命令及参数说明见下图：

逃逸名称	命令	参数1	参数2	参数3	备注
IP分片	ip_frag	size (分片大小)			
IP重叠	ip_overlap	size (分片大小)	overlap(片重叠大小)	model(重叠规则)	model为0/1，0表示后一个覆盖前一个
TCP分段	tcp_frag	size (分段大小)			
TCP段重叠	tcp_overlap	size (分段大小)	overlap(段重叠大小)	model(重叠规则)	model为0/1，0表示后一个覆盖前一个
修改TTL	modify_ttl	size(新的TTL值)			
修改TCP选项	modify_tcp_flags	flag(新的tcp标志位)			flag范围为(syn，ack，fin，rst，psh)
修改MSS	modify_mss	size(新的MSS值)			
修改wscale	modify_wscale	size(新的wsacle值)			

表 3-4 配置文件的配置规则

举个例子，对于下面的这个配置文件:

ip_frag 4

modify_ttl 64

modify_tcp_flags syn

它表示的含义就是对每个流经的数据包先进行 IP 分片，分片大小为 4，然后修改 TTL 为 64，最后修改 TCP 选项为 syn。

3.2.4 SAPP 交互

SAPP 平台是一个大规模实时流处理平台，因为其具有全协议栈解析能力，所以实质上，SAPP 对该系统屏蔽了协议栈的细节。对于其上层的服务，SAPP 提供了一个封装好的接口 API 供调用。这样的好处是在实现该逃逸系统时，对于收发数据直接调用 SAPP 的接口就行，从而把重心集中在逃逸处理上。

在接收流量方面，SAPP 提供了多种接口，包括 IP 层流量（经过碎片重组之后），TCP 层流量（经过 TCP 流还原之后），甚至 SAPP 还提供了以太网层的原始流量。这我们这个系统中，我们拿到的是 IP 层的流量。IP 层的流量入口函数如下：

char business_ip_entry(struct streaminfo*f_stream, unsigned char routedir, int thread_seq, const void* entry_ip_pkt);

其中，fstream：当前流的信息

routedir：上下行方向

thread_seq：当前线程

entry_ip_pkt: 指向一个 IP 包的指针，也就是我们要处理的包

在发送数据包方面：SAPP 提供了各种封装好的发包函数，包括发送 IP 包，TCP 包，以太网帧等，这里我们采用的是发送 IP 包。SAPP 提供的发送 IP 包的函数如下：

MESA_sendpacket_iplayer(int thread_index,const char* ip_pkt,int buf_len,int dir);

其中：thread_index: 当前线程

ip_pkt：要发送的 IP 包的指针

buf_len: 要发送的 IP 包的长度

dir: 上下行方向

3.3 逃逸方法设计

该系统需要实现 8 种逃逸方法，具体见表 3-3。下面针对每种逃逸方法的设计思想加以说明。

3.3.1 IP 分片

IP 分片是一种最常见，最传统的逃逸方法，它的主要原理是通过将 IP 数据包分成很小的碎片，从而将恶意代码所具有的特征消除。因为大部分 IDS 都是基于特征匹配的，所以这样就可以逃过 IDS 的检测。下图展示了 IP 分片的原理。

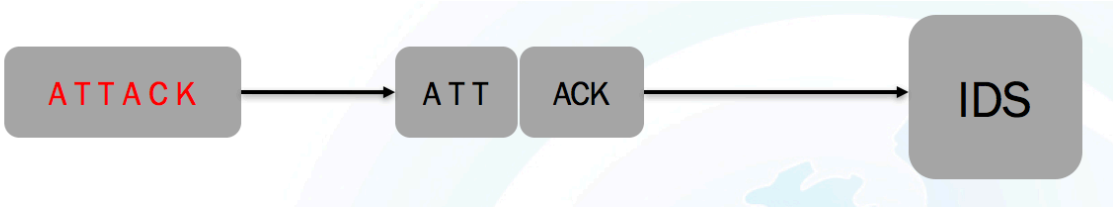


图 3-5 IP 分片的过程

那么怎么才能实现 IP 分片呢，这是我们就需要对 IP 头结构有所了解。下图是一个 IP 报文的结构。

图 3-6 IP 报文格式

在 Linux 的 tcp.h 中，定义的 ip 报文头的结构如下：

```
struct ip {
    u_int    ip_hl:4,          //头部长度
             ip_v:4;           //版本
    u_char   ip_tos;           //服务类型
    u_short     ip_len;        //IP 包总长度
    u_short     ip_id;         //16 位标识
    u_short     ip_off;        // 偏移量
    u_char      ip_ttl;        // 生存周期
    u_char      ip_p;          // 8 位协议
    u_short     ip_sum;        // 校验和
    struct      in_addr ip_src,ip_dst;  //源地址和目标地址
};
```

在 IP 头部有 4 个字节是用于分片的，分别是 16 位标识，13 位偏移量，以及 3 个标志位 DF（Don't Fragment），MF（More Fragment），RF。也就是对应 ip 头部的 ip_id 和 ip_off 字段。其中，属于同一个分片的 16 位标识是相同的。13 位偏移量，指的是分片后的报文相对于原始报文的偏移量，以 8 字节为一个单位。至于另外 3 个标志位，DF 表示不分片，如果将这一位置为 1，IP 层将不对该报文进行分片。MF 表示有更多的分片，除了最后一个分片，其他分片的 MF 值都为 0。RF 是保留位，暂时没有用到。

所以我们可以得到下面的这个对应关系：

DF	MF	分片状态
0	0	最后一个分片
0	1	不是最后一个分片
1	0	不分片
1	1	不分片

表 3-7 IP 分片的状态图

需要分片的部分是 IP 包的 payload，即负载部分，对于每个新建的 IP 分片，我们需要填充的字段包括 ip_len(ip 包总长度)，ip_off(偏移量)，ip_sum(IP 头部校验和)。其他字段的与分片前的 IP 头部字段相同即可。

ip_len：ip 包总长度，这个长度包括头部和负载部分。一般来说，头部长度都是 20 字节，所以 ip_len 就是负载+20。但是这样做不太安全，因为在有些情况下，有些 ip 包头部还存在 ip 选项。所以正确的做法是 ip_len=payload+ip_hl*4。因为 IP 头部有个字段 ip_hl，表示 IP 头部的长度，注意乘以 4，因为 ip_hl 是以 4 个字节为一个单位计算的。

ip_off：ip 包的偏移量，指的是 ip 分片针对分片前报文的偏移量，8 个字节为一个单位，所以在对原始报文进行分片后，需要将偏移量除以 8，然后填充到 IP 头部。需要注意的是，ip 偏移量是 ip_off 的后 13 位。ip_off 的前 3

位是分片的标志位。如上表所示，对于不是最后一个分片，需要将 DF 和 MF 都置为 0，对于最后一个分片，需要将 DF 置为 0，MF 置为 1。

ip_sum：这个指的是 ip 头部的校验和，下面是 ip 头部校验和的算法：

1)：将 ip 头的校验和字段清零

2)：将 IP 头部转换成 2 进制

3)：以 16 位（2 个字节）为一个单位，对 IP 头部进行反码求和

4)：得到的结果即为 IP 校验和

下面是计算校验和的代码：

```
SHORT checksum(USHORT* buffer, int size)
{
    unsigned long cksum = 0;
    while(size>1)
    {
        cksum += *buffer++;
        size -= sizeof(USHORT);
    }
    if(size)
    {
        cksum += *(UCHAR*)buffer;
    }
    cksum = (cksum>>16) + (cksum&0xffff);
    cksum += (cksum>>16);
    return (USHORT)(~cksum);
}
```

3.3.2 IP 重叠

IP 重叠是在 IP 分片的基础上进行的，指的是 IP 的分片之间有重叠。IP 分片虽然可以逃逸检测，但是现在一般的 IDS 都会对 IP 分片进行重组，还原出原始报文。在这种情况下，IP 分片就失去了用武之地，就需要 IP 重叠出马了。IP 重叠利用了主机和 IDS 在碎片重组时的处理不一致的漏洞。一般来说，在遇到 IP 重叠的问题时，一般有两种解决方案：用后一个包覆盖前一个包，或者用前一个包覆盖后一个包。下面这张图形象解释了 IP 重叠的原理：

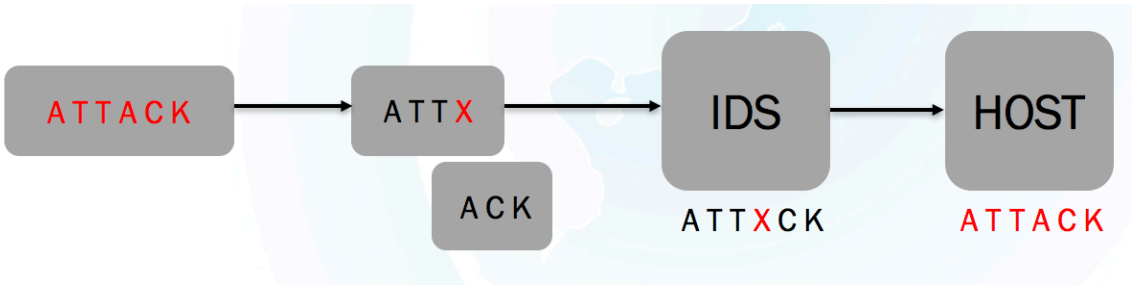


表 3-8 IP 重叠示意图

如图所示，现在有一个恶意代码称作 ATTACK，现在被分成了两个分片 ATTX，ACK。并且在第 4 个位置产生了 IP 重叠，在第一个分片中是 X，在第二个分片中是 A，那么主机或者 IDS 在重组这个 IP 分片时就产生了二义性，不同的操作系统或者协议栈对这种情况的处理方式不一致。如果我们假设 IDS 的处理方式是前一个包覆盖后一个包，那么 IP 包被重组成为 ATTXCK，这个包并不是原始的报文，不含攻击特征，所以能够逃过 IDS 的检测。等 IP 分片到达了目标主机之后，目标主机的重组方式是后一个包覆盖前一个包，那么 IP 包就会成功被重组为 ATTACK。这是原始的恶意代码，于是目标主机成功被攻击但是 IDS 却没有检测出来。

这种攻击方式实际上就是利用了 IP 重叠时处理的二义性。通常来说，一个 IDS 后面有很多台主机，这些主机的擦作系统和协议栈不可能完全相同。所以 IDS 和主机处理方式不同的情况总是会存在。也就是说，攻击者总是可以利用 IP 重叠进行攻击。

因为 IP 重叠和 IP 分片的区别仅仅在于 IP 重叠的 payload 部分有重叠，剩下的 IP 头部的生成方式和 IP 分片完全一样。所以在设计系统的 IP 重叠方法时，只需要把重叠的部分数据生成好，然后直接调用 IP 分片模块即可。举个例子，如果一个 IP 数据包为 ABCDEF，分片大小为 4，重叠大小为 2，覆盖方式为后一个包覆盖前一个包。那么我们先把数据部分转换为 AB**CD**EF**。然后调用 IP 分片，就分成了 3 个分片 AB**，CD**，EF**

3.3.3 TCP 分段

与 IP 分片相似，TCP 分段的原理也是通过将 TCP 的负载分段，从而使恶意代码原来的特征被打散，然后逃过 IDS 的检测。IP 分片和 TCP 的不同就在于 IP 分片是在网络层分片，TCP 分段是在传输层进行分段。因为 TCP 协议比较复杂，所以 TCP 分段需要考虑的东西也就比较多。

要对 TCP 进行分段，那么就需要对 TCP 的头部有所了解。如下图所示，下图是一个 TCP 的头部结构图。

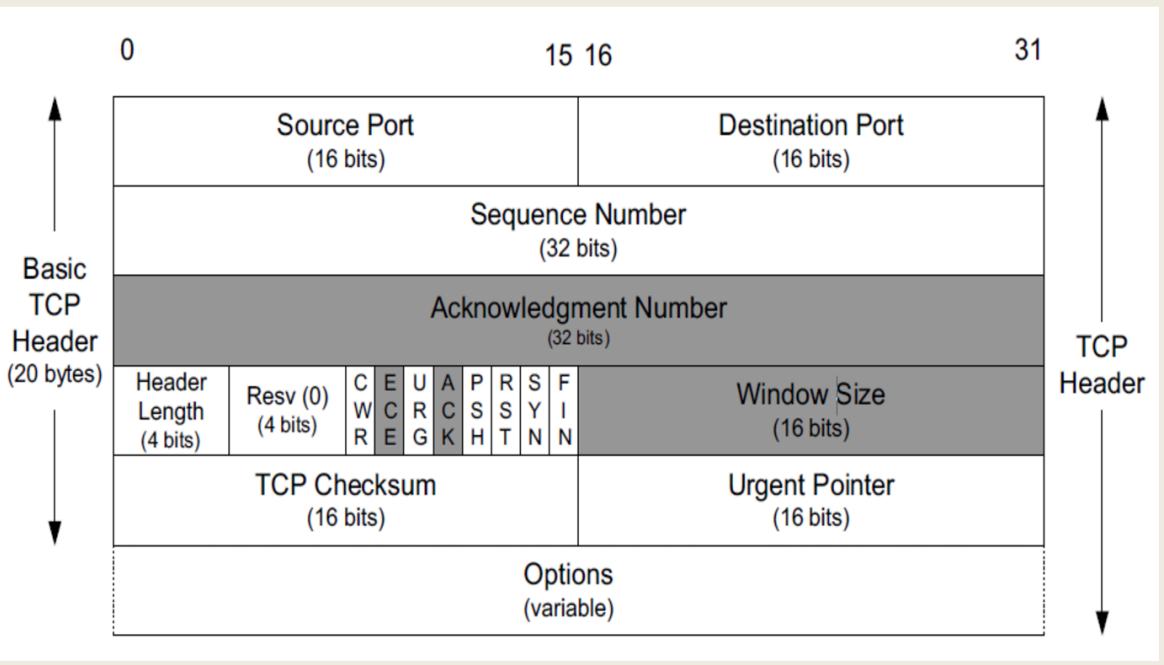


表 3-8 IP 重叠示意图

如上图所示，TCP 的基本头部含有 20 字节，不含额外的 TCP 选项。TCP 头部包含源端口，目的端口，seq 值，ack 值，头部长度，TCP 标志位，窗口大小，TCP 头部校验和，紧急指针等。

在 Linux 的 tcp.h 中，tcp 的定义如下：

因为 TCP 是基于流的，可靠的网络传输协议。所以，在对 TCP 进行分段

```
struct    tcphdr {
    unsigned short     th_sport;      //源端口
    unsigned short     th_dport;      //目的端口
    unsigned int    th_seq;           //seq值
    unsigned int    th_ack;           //ack值
    unsigned int    th_off:4,         // tcp头部长度
            th_x2:4;
    unsigned char th_flags;           // tcp标志位
    unsigned short     th_win;         //窗口大小
    unsigned short     th_sum;         // tcp校验和
    unsigned short     th_urp;         //紧急指针
};
```

之后，需要更改的值只有 seq 值和 TCP 校验和，因为 seq 值就可以代表一个 TCP 段，下面对这两个字段加以说明：

th_seq：是 TCP 的序列号，是 32 位整数。TCP 分段后，新的 TCP 报文的序列号是 TCP 分段前的序列号加上该段的偏移。

th_sum：是 TCP 的校验和，TCP 校验和是一层简单的校验，用来确保 TCP 报文没有被修改，TCP 校验和由发送方填充，接收方校验，如果 TCP 校验和不正确，则该 TCP 包会被直接丢弃。

和 IP 层的校验和不同，TCP 层的校验和 TCP 的校验和算法如下：

1) 把 TCP 校验和字段置为 0
2) 把整个 TCP 报文段转换成 2 进制

3) 将 2 进制数据以 16 位为单位反码求和

4) 求得的和即为 TCP 校验和

下面是求 TCP 校验和的代码：

```
void    modify_tcp_checksum(struct iphdr* ip_pkt){
      int ip_hdr_len=ip_pkt->ihl*4;
      int ip_tot_len=ntohs(ip_pkt->tot_len);
      int size=ip_tot_len-ip_hdr_len;
      u_int16_t* buffer=(u_int16_t*)(((char*)ip_pkt)+ip_hdr_len);
      uint32_t saddr=ip_pkt->saddr;
      uint32_t daddr=ip_pkt->daddr;
      ((struct tcphdr*)buffer)->check=0;
       uint32_t sum;
       uint16_t *w;
       int nleft;
       sum = 0;
       nleft = size;
       w = buffer;
       while (nleft > 1){
             sum += *w++;
             --nleft;
            --nleft;
       }
       if (nleft)
             sum += *w & ntohs(0xFF00);
       sum += (saddr & 0x0000FFFF) + (saddr >> 16);
       sum += (daddr & 0x0000FFFF) + (daddr >> 16);
       sum += htons(size);
       sum += htons(IPPROTO_TCP);
       sum = (sum >> 16) + (sum & 0xFFFF);
       sum += (sum >> 16);
       sum = ~sum;
       ((struct tcphdr*)buffer)->check=(uint16_t)sum;
}
```

需要注意的是，在修改完 TCP 段的信息之后，还要修改 IP 层的 tot_len 和校验和，因为修改 TCP 段的信息也影响了 IP 层的这两个字段。

3.3.4 TCP 段重叠

TCP 段重叠和 IP 包重叠原理一样，区别就在于一个在网络层，一个在传输层。这里不再详述。

3.3.5 修改 TTL

TTL（time to live），指的是 IP 包被路由器丢弃之前允许通过的最大网段数量，TTL 的单位是跳。IP 包每次经过一个路由器，TTL 的值就会减一。如果到达某个路由器的 IP 包的 TTL 为 0，那么这个包就会被丢弃。通过巧妙地修改 TTL 的值可以实现巧妙地实现逃逸。

下面这个图形象地展示了如何通过修改 TTL 的值来逃逸 IDS 的检测。

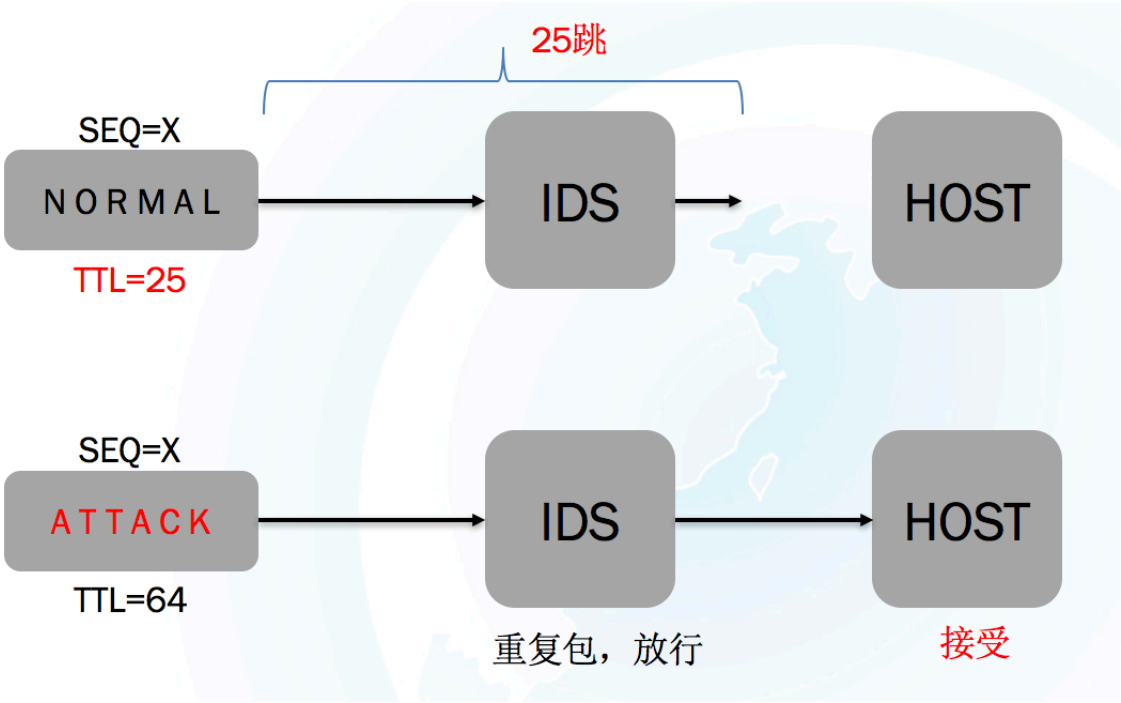


表 3-9 修改 TTL 实现逃逸的原理图

如上图所示，从攻击者到目标主机中间一共有 64 跳，现在有一个恶意代码 ATTACK，如果直接发送 ATTACK 给目标主机一定会被 IDS 检测出来。这个时候如果修改 TTL 就可以巧妙地实现逃逸。

我们假设 ATTACK 攻击包的序列号 seq=x，生存时间 TTL=64。在发送这个 ATTACK 攻击包之前，我们可以先构造一个正常的 TCP 包 NORMAL。这个 NORMAL 包的序列号 seq=x，生存时间 TTL=25。然后我们先发送这个 NORMAL 包，由于这是一个正常的包，所以一定不会被 IDS 检测出来，但是同时，由于这个 NORMAL 包的 TTL 只有 25，但是从发送方到目标主机的 TTL 为 64，所以 NORMAL 包在到达目标主机之前就被丢弃了。这时我们再发送 ATTACK 包，当 ATTACK 包到达 IDS 的时候，IDS 发现 seq=x 这个包之前已经检测过（NORMAL）包，所以会直接放行。因此，通过巧妙地修改 TTL 就可以实现逃逸 IDS 的检测。

这种逃逸方法的关键在于如何选择合适的 TTL，使得 IP 包能够到达 IDS 却无法到达目标主机。对此，我们可以不停地修改 TTL 来测量出发点到 IDS 的跳数。

值得注意的是，修改 TTL 本质上还是构造一个新的 IP 包，所以修改完 TTL 之后，还是要注意修改 IP 层的 tot_len，校验和，TCP 层的校验，使其符合规则。

3.3.6 修改 TCP 标志位

TCP 标志位是 TCP 头部最重要的一个字段。它代表着这个 TCP 报文的属性。TCP 报文一共拥有 5 个 TCP 标志位，SYN，ACK，FIN，PSH，RST。下面对这几个标志位加以说明：

SYN：同步位，用于 TCP 建立连接时同步序号

ACK：确认位，表示确认收到一个 TCP 报文

FIN： 终止位，用于在传输结束后断开连接

PSH：推送位，表示在传输数据

RST：复位位，表示重置一个 TCP 连接

如果想要修改某个 TCP 报文的标志位，只需要把该标志位的值置为 1 即可。需要注意的是，修改 TCP 的标志位本质上还是在构造一个新的 IP 包，所以修改完 TCP 的标志位之后，还需要修改 IP 层的 tot_len，校验和，TCP 层的校验和，使其符合规则。

3.3.7 修改 MSS

MSS: Maximum Segment Size 最大分段大小，指的是 TCP 的报文段的最大长度。在介绍 MSS 之前，需要先了解一下 MTU（Maximum Transport Unit），最大传输单元，这是由以太网的帧来决定的，一般为 1500。而 MSS 的大小通常在 TCP 建立连接时确定。因为 TCP 和 IP 的包头都是 20 个字节，所以 MSS 的大小一般为 1460。

如果想要修改某个 TCP 报文的 MSS，只需要把 TCP 报文对应的 MSS 字段改为需要的值即可。需要注意的是，修改 TCP 的标志位本质上还是在构造一个新的 IP 包，所以修改完 TCP 的标志位之后，还需要修改 IP 层的 tot_len，校验和，TCP 层的校验和，使其符合规则。

3.3.8 修改 wscale

wscale 是窗口扩大因子选项，在介绍窗口扩大因子选项之前，必须先说下一窗口大小，在 TCP 传输过程中为了更快的传输数据，采用了滑动窗口协议，发送窗口的大小代表着接收/发送数据包的速度大小。而窗口扩大因子就代表着窗口大小扩大的倍数。举个例子，如果原来窗口大小是 w，窗口扩大因子是 x，那么新的窗口大小就是 w*2^x。因为 wscale 只有一个字节，所以 wscale 的最大值为 255。

需要注意的是，窗口扩大因子只能携带在 SYN/ACK 包，而且需要双方都开启才会生效。

另外，修改 TCP 的 wscale 本质上还是在构造一个新的 IP 包，所以修改完 TCP 的标志位之后，还需要修改 IP 层的 tot_len，校验和，TCP 层的校验和，使其符合规则。

第 4 章 逃逸系统的实现

基于第三章的逃逸系统的设计，本文实现了一个基于 SAPP 的高级逃逸系统，系统包含 8 种逃逸方法，支持用户配置逃逸命令组合。同时，由于系统采取了模块化设计，所以支持动态添加逃逸方法，可拓展性很强。

整个系统用 C 语言编写，代码行数超过 1500 行，共包含 19 个文件。下面对每个文件的功能加以说明。

4.1 代码文件说明

4.1.1 AET_test.c

这是整个系统的入口文件，共包含三个函数：

AET_init：插件初始化函数，平台第一次加载插件时执行这个函

AET_destory：插件析构函数，插件停止运行执行这个函数

business_ip_entry：插件从平台获取流量的入口

如下图所示：

```
/* 插件初始化*/
int AET_init(){
      printf("AET: succeed to init\n");
      init_command_set_system(); //初始化逃逸命令
      init_command_set_user();
}
/* 插件析构函数*/
void AET_destroy(){
      printf("AET: succeed to destroy\n");
}
/* 获取流量入口*/
char business_ip_entry(struct streaminfo*f_stream,unsigned
char routedir,int thread_seq,const void* entry_ip_pkt){
      printf("Succeed: businiss_ip_entry called");
      exec_command_set(&send_queue); //执行逃逸命令
      return APP_STATE_GIVEME;
}
```

4.1.2 makefile

makefile 指定了系统中各种文件之间的编译规则和依赖，由于最终该系统是以插件的形式运行，所以需要把系统编译为 .so 动态链接库的形式。makefile 的内容如下面所示：

```
cc=gcc
objects = AET_test.o module_ip_frag.o module_ip_overlap.o
AET_test.so : $(objects)
	cc -g -Wall -shared -o AET_test.so $(objects)
AET_test.o : AET_test.c
	cc -g -Wall -fPIC -c AET_test.c
module_ip_frag.o : module_ip_frag.c
	cc -g -Wall -fPIC -c module_ip_frag.c
module_ip_overlap.o : module_ip_overlap.c
	cc -g -Wall -fPIC -c module_ip_overlap.c
.PHONY : clean
clean :
	rm AET_test.so $(objects)
```

需要注意的是，上面的 makefile 并不是完整的，这里这是截取部分 makefile 来说明一下这个 makefile 的作用。

4.1.3 configuer.c

这个文件是控制器文件，它负责逃逸处理的整个流程，包括读取配置文件，获取输入流量，执行逃逸命令，发送处理后的数据包等，是整个程序最重要的部分。该文件一共包含下面几个主要的函数。

```
/* 初始化系统命令集，即系统支持的逃逸命令 */
void init_command_set_system();

/* 初始化用户命令集，即读取用户配置的逃逸命令 */
void init_command_set_user();

/* 依次执行配置里面的逃逸命令 */
void exec_command_set();
```

4.1.4 Utils.c

这是系统所需的功能函数集合，主要把每个模块都需要用到的通用函数提取出来放在了 Utils.c 文件中。该文件一共包含下面的几个函数。

```
/*  复制另一个 ip 包的头部  */
void copy_ip_header();

/*  复制另一个 tcp 包的头部  */
void copy_tcp_header();

/*  修改 ip 包的校验和  */
void modify_ip_checksum ();

/*  修改 tcp 包的校验和  */
void modify_tcp_checksum();

/*  自定义的字符串复制函数  */
void mystrcpy();
```

4.1.5 module_ip_frag.c

IP 分片模块，输入待分片的包，将分片结果存入 send_queue 队列中。具体的程序请阅读源代码。

4.1.6 module_ip_overlap.c

IP 重叠模块，输入待处理的包，将经过 IP 重叠包处理后的结果存入 send_queue 队列中，具体的程序请阅读源代码。

4.1.7 module_tcp_segment.c

TCP 分段模块，输入待处理的包，将经过 TCP 分段处理后的结果存入 send_queue 队列中，具体的程序请阅读源代码。

4.1.8 module_tcp_segment.c

TCP 段重叠模块，输入待处理的包，将经过 TCP 段重叠处理后的结果存入 send_queue 队列中，具体的程序请阅读源代码。

4.1.9 module_modify_ttl.c

修改 IP 的 TTL 模块，输入待处理的包，将修改过 TTL 的包存入 send_queue 队列中，具体的程序请阅读源代码。

4.1.10 module_modify_tcp_flags.c

修改 TCP 包头的的标志位模块，输入待处理的包，将修改过 TCP 包头标志位的包存入 send_queue 队列中，具体的程序请阅读源代码。

4.1.11 module_modify_mss.c

修改 TCP 的包头的最大段长度模块，输入待处理的包，将修改过 MSS 的包存入 send_queue 队列中，具体的程序请阅读源代码。

4.1.12 module_modify_wscale.c

修改 TCP 的窗口因子选项模块，输入待处理的包，将修改过 wscale 的包存入 send_queue 队列中，具体的程序请阅读源代码。

4.1.13 module_send_packet.c

发送 send_queue 中处理后的包模块，具体的程序见源代码。

4.1.14 各种头文件

为了各个模块之间解耦，更好地进行模块化设计，系统将各个模块对应的头文件都单独建立出来。系统的头文件如下：

AET_test.h：AET_test.c 的头文件，包括了 AET_test.c 里面函数的定义

configure.h：configure.c 的头文件，包括了 configure.c 里面函数的定义

module.h：module.c 的头文件，包括了 module.c 里面函数的定义

utils.h：utils.c 的头文件，包括了 utils.c 里面函数的定义

第 5 章 实验结果及分析

由于本实验主要是对捕获的 TCP/IP 包进行逃逸处理，所以实验结果主要是通过捕包来观察。因为实验环境是 linux，所以实验采用了 tcpdump 进行捕包。同时，由于该系统和开源软件 Fragroute 的功能相似，所以实验中除了自己对比之外，还选择了和 Fragroute 进行对比实验。

1.1. 实验步骤

由于实验需要模拟真实的网络的网络通信，所以实验在内网进行，实验一共选取了 3 台主机进行，分别为 A，B，C。其中 A，B 两台主机负责通信，C 主机上运行着 SAPP 平台，负责逃逸处理，A 发送 B 主机的包，经过 C 主机的逃逸处理才发往 B。实验中保证 A，B 直接的流量一定经过 C。如下图所示：

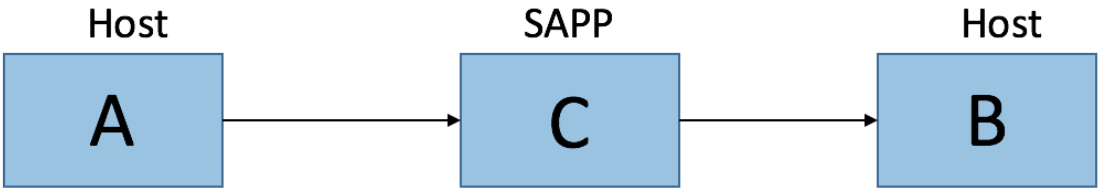


图 3-10 实验连接示意图

在本实验中，建立网络通信利用的是 nc 命令，捕包利用的 tcpdump 命令，关于这两个命令的简单说明如下：nc 命令是 netcat 命令的简写，有着网络界的瑞士军刀的美誉，是一个简单，可靠的网络工具。它可以实现任意 TCP/UDP 端口的侦听，同时可以作为客户端发起 TCP/UDP 连接，所以利用 nc 命令可以快速地建立一个 TCP/UDP 连接。tcpdump 是 linux 一个功能强大的网络捕包工具，底层用的是 pcap 捕包实现，并且支持网络层，协议，端口，主机等条件的过滤，还支持 or，not，not 等逻辑语句。本实验中利用 tcpdump 进行网络捕包。

实验中所需要的步骤如下：

```
make &&make install      //编译安装 SAPP 平台和逃逸
```

```
./sapp      //运行 SAPP 平台
```

```
nc -t -l 9999    //监听 TCP 的 9999 端口
```

```
nc -t 10.0.6.86 9999    //与 86 机器建立 TCP 连接
```

```
tcpdump -i eth1 'tcp and port 9999 and host 10.0.9.86'    //捕包
```

1.2. 实验结果

该系统一共实现了 8 种逃逸方法，在该实验中，对这 8 种逃逸方法都进行了实验，共捕获网络包 16 个，实验结果如下：

a) IP 分片前：

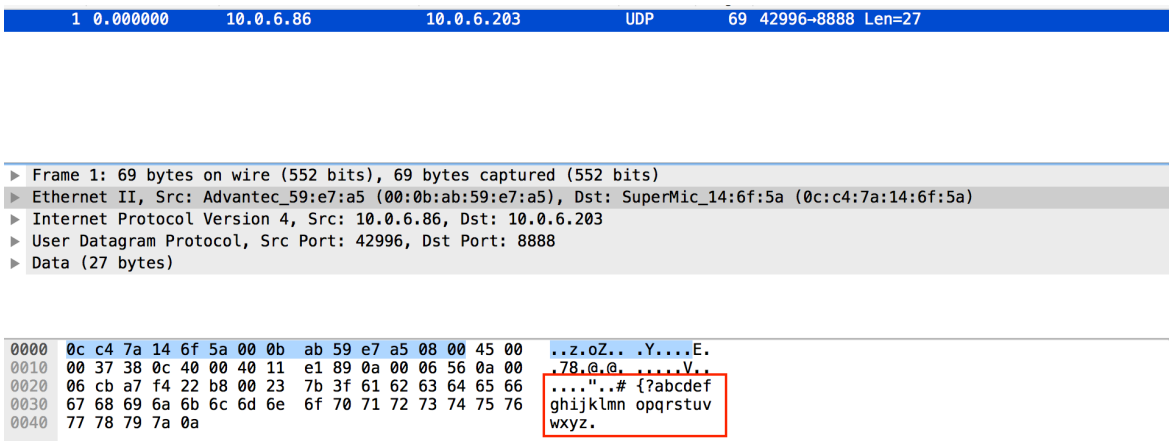


图 3-11 IP 分片前

a1) IP 分片后：

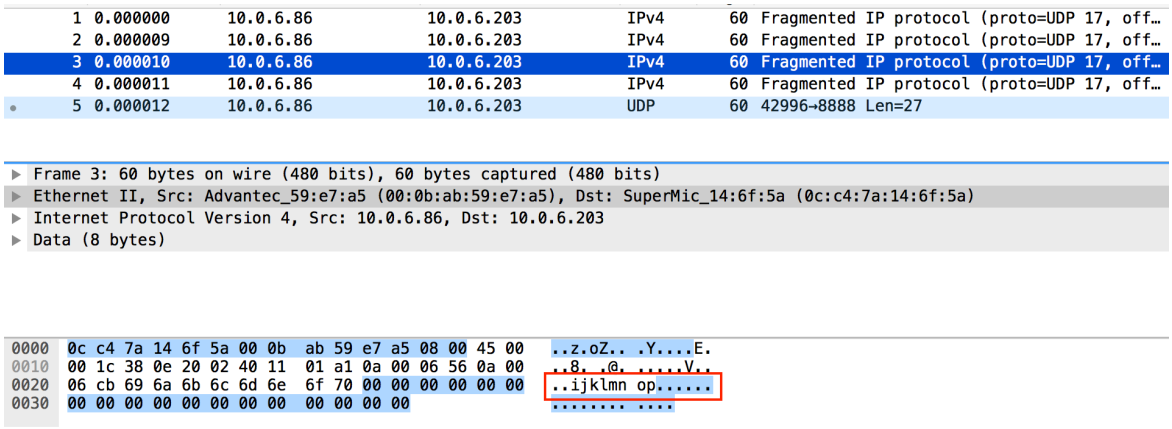


图 3-12 IP 分片后

b) TCP 分段前：

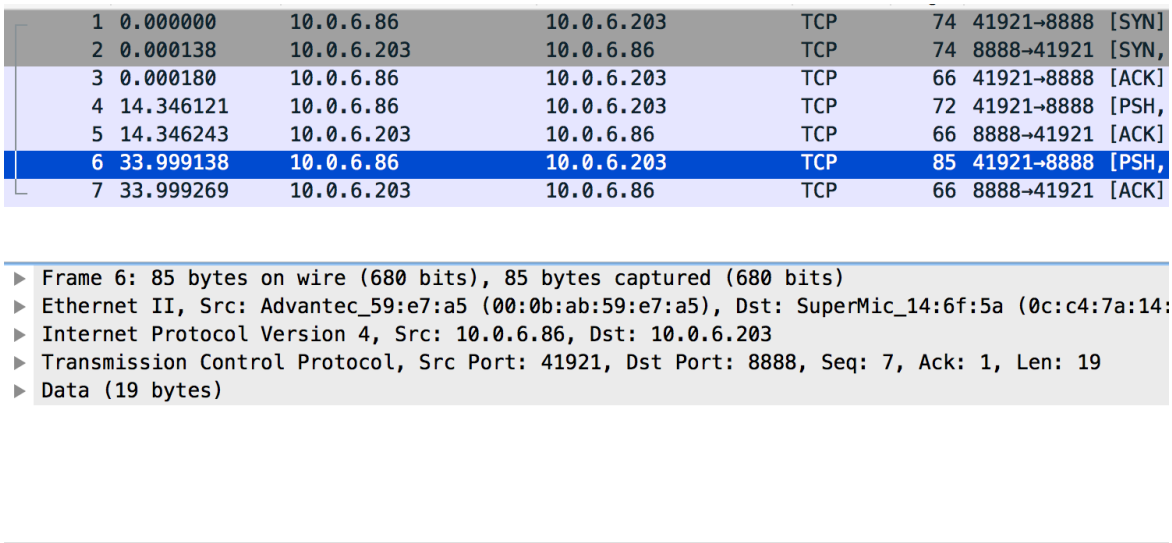


图 3-13 TCP 分段前

b1) TCP 分段后：

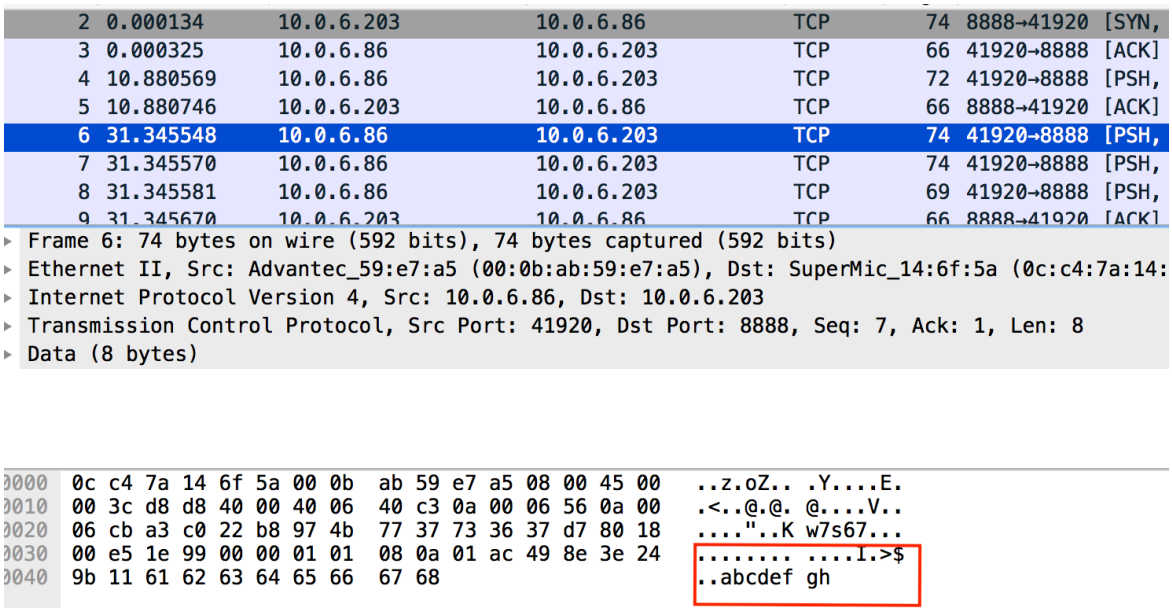


图 3-14 TCP 分段后

c) 修改 TTL 前：

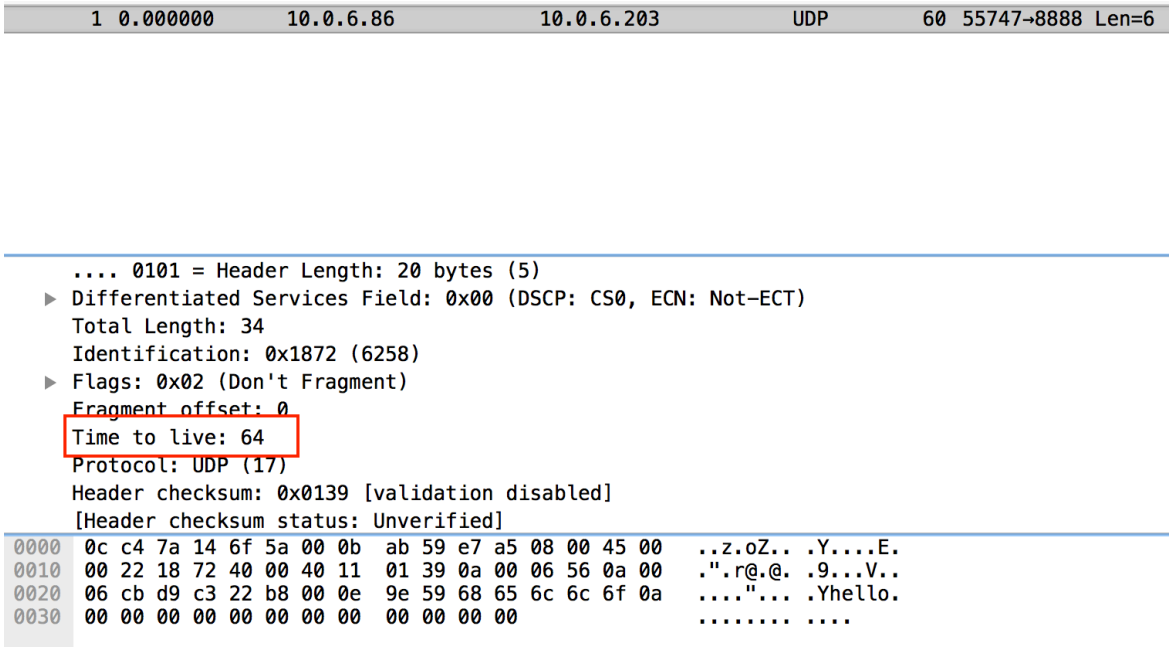


图 3-15 修改 TTL 前

c1) 修改 TTL 后：

```
    1 0.000000    10.0.6.86     10.0.6.203    UDP      60 22126→18788 [BAD UDP LEN
    2 0.000015    10.0.6.86     10.0.6.203    UDP      60 43883→8888 Len=6
```

```
▶ Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
▶ Ethernet II, Src: Advantec_59:e7:a5 (00:0b:ab:59:e7:a5), Dst: SuperMic_14:6f:5a (0c:c4:7a:14:6f:5a)
▼ Internet Protocol Version 4, Src: 10.0.6.86, Dst: 10.0.6.203
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 34
    Identification: 0x6184 (24964)
  ▶ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
0000  0c c4 7a 14 6f 5a 00 0b  ab 59 e7 a5 08 00 45 00   ..z.oZ.. .Y....E.
0010  00 22 61 84 40 00 1e 11  da 26 0a 00 06 56 0a 00   ."a.@... .&...V..
0020  06 cb 56 6e 49 64 54 50  4a 76 56 75 6c 33 73 5a   ..VnIdTP JvVul3sZ
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

图 3-16 修改 TTL 之后

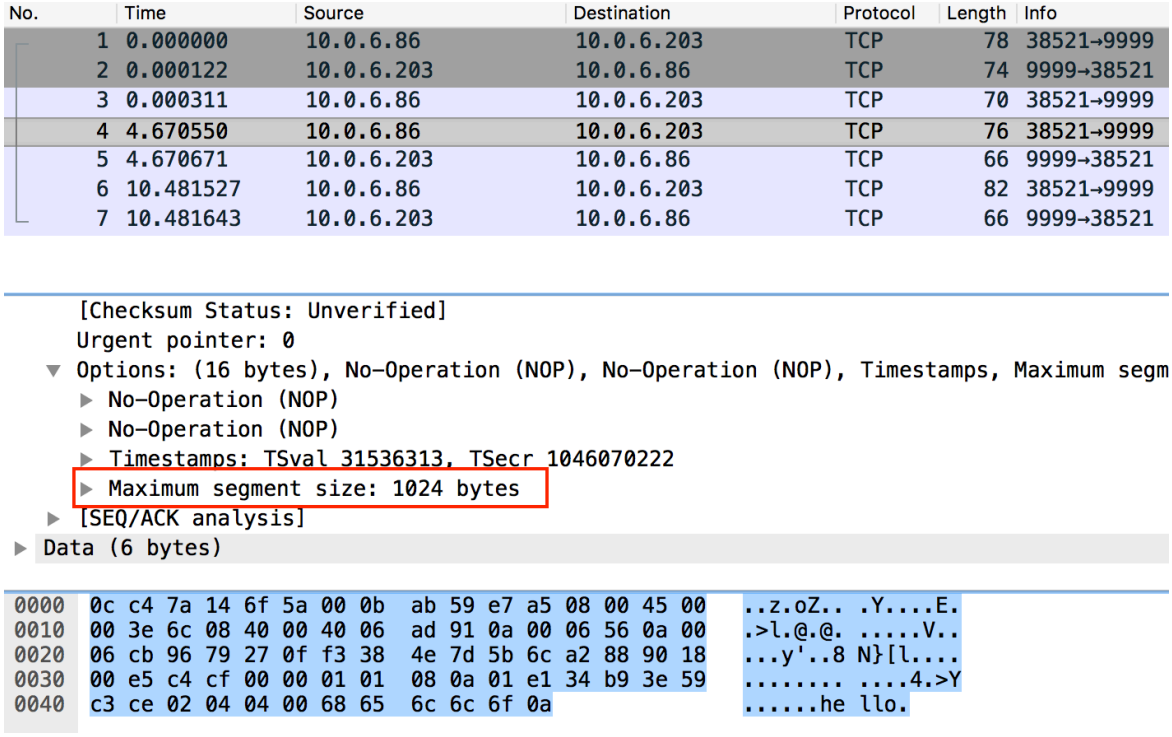
d) 修改 MSS 前：

```
    1 0.000000    10.0.6.86     10.0.6.203    TCP      74 38520→9999 [SYN]
    2 0.000127    10.0.6.203    10.0.6.86     TCP      74 9999→38520 [SYN,
    3 0.000156    10.0.6.86     10.0.6.203    TCP      66 38520→9999 [ACK]
    4 11.635831   10.0.6.86     10.0.6.203    TCP      72 38520→9999 [PSH,
    5 11.635949   10.0.6.203    10.0.6.86     TCP      66 9999→38520 [ACK]
    6 17.159811   10.0.6.86     10.0.6.203    TCP      78 38520→9999 [PSH,
    7 17.159959   10.0.6.203    10.0.6.86     TCP      66 9999→38520 [ACK]
```

```
    Window size value: 14600
    [Calculated window size: 14600]
    Checksum: 0xc75d [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▼ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP),
    ▶ Maximum segment size: 1460 bytes
    ▶ TCP SACK Permitted Option: True
    ▶ Timestamps: TSval 31433459, TSecr 0
    ▶ No-Operation (NOP)
0000  0c c4 7a 14 6f 5a 00 0b  ab 59 e7 a5 08 00 45 00   ..z.oZ.. .Y....E.
0010  00 3c 50 35 40 00 40 06  c9 66 0a 00 06 56 0a 00   .<P5@.@. .f...V..
0020  06 cb 96 78 27 0f d1 59  f2 c6 00 00 00 00 a0 02   ...x'..Y ........
0030  39 08 c7 5d 00 00 02 04  05 b4 04 02 08 0a 01 df   9..]....  ........
0040  a2 f3 00 00 00 00 01 03  03 06                     ........ ..
```

图 3-17 修改 MSS 前

d1) 修改 MSS 后



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.6.86	10.0.6.203	TCP	78	38521→9999
2	0.000122	10.0.6.203	10.0.6.86	TCP	74	9999→38521
3	0.000311	10.0.6.86	10.0.6.203	TCP	70	38521→9999
4	4.670550	10.0.6.86	10.0.6.203	TCP	76	38521→9999
5	4.670671	10.0.6.203	10.0.6.86	TCP	66	9999→38521
6	10.481527	10.0.6.86	10.0.6.203	TCP	82	38521→9999
7	10.481643	10.0.6.203	10.0.6.86	TCP	66	9999→38521

图 3-18 修改 MSS 后

1.3. 与 Fragroute 对比

由于该系统的功能和 Fragroute 的功能类似，所以为了更好的说明实验效果，本实验还和开源软件 Fragroute 进行了对比实验。

Fragroute 是一块开源的 TCP/IP 层的逃逸工具。2002 年 3 月，http://www.monkey.org/~dugsong/发布了一个工具 Fragroute，在入侵检测领域引起了很大的震动，有关 Fragroute 的讨论成了 3-4 月的一个热点。

Fragroute 能够截取，修改，和重写发送出去的报文，实现了大部分在 TCP/IP 层的逃逸技术。包括 TCP 分段，IP 分片，IP 重叠，修改 TCP 和 IP 头部的选项等。

Fragroute 是一个运行在本地主机的逃逸工具，它运行在发送者的机器上，在报文真正发出之前就完成了逃逸。Fragroute 的原理如下：

i) 在发送发的路由表中增加一条路由，使得所有发往目标主机的报文都被重定向到本机的 lo 端口

ii) 监听本机的 lo 端口，并且过滤出发往目标主机的包并且进行 pcap 捕包。

iii) 读取用户的配置文件，对捕获的报文进行逃逸处理。

Iv) 将逃逸处理后的报文发往目标主机。

为了和 Fragroute 进行对比试验，本文安装了 Fragroute 并且利用其进行和本系统进行相同的逃逸处理。得到的结论如下：

1） 在功能上，二者没有什么区别，Fragroute 能够实现的功能该系统均可以实现。都能进行 TCP/IP 层的逃逸处理。

2） 在兼容性上，由于 Fragroute 是 2002 年发布的，距今已经 15 年了，所以难免会有一些兼容性的问题，测试发现，在 centos 7 无法兼容 Fragroute。而我们的系统则不存在这个问题。

3） 在普适性上，由于 Fragroute 是开源工具，其依赖于 libpcap，libevent，libdnet 等第三方库。而 Fragroute 的底层服务则依赖于 SAPP 平台提供的接口。而且是以插件的形式运行在 SAPP 平台上，所以是针对特定平台的特定软件。而 Fragroute 则不存在这个问题，所以在普适性上，Fragroute 优于该系统。

1.4. 实验结果分析

该系统一共实现了 8 种逃逸方法，且支持逃逸系统的组合，在实验过程中发现，经过该系统的逃逸处理之后，确实可以达到预期的效果。

从本质上来说，该系统只是实现了一个 TCP/IP 层的逃逸方法的工具集或者说框架，具体逃逸的效果如何取决于逃逸方法的合适组合，逃逸参数的合适选择。

另外，正如上文所说，该系统以插件的形式运行在 SAPP 平台上，底层的服务都依赖于 SAPP 的接口。所以是针对特定环境，特定平台的逃逸工具。从普适性上来讲，这是该系统的一个缺陷。

第六章　未来工作

1.1 在 IDS 和防火墙上进行测试

由于逃逸技术最终是针对 IDS 或者防火墙的，所以在真实的 IDS 或者防火墙上进行测试是必须的。而在该实验中，用的是网络捕包的方法来模拟 IDS 的检测，这是不严谨的。所以未来计划在线下搭建一个真实的 IDS（snort），自己配置对应的规则，通过观察 IDS 是否报警来判断出逃逸处理是否有效果。更进一步，甚至可以

在防火墙进行测试，在线下搭建一个防火墙，然后自己配置过滤规则，通过观察恶意代码能否通过防火墙来检测逃逸效果。

1.2 增加更多的逃逸方法

IDS 和逃逸方法之间的战争从未停止。随着 IDS 的不断进步，很多传统的逃逸方法已经很难逃过 IDS 的检测，另一方面，新的逃逸方法也层出不穷。在这种情况下，该系统仅有的 8 种逃逸方法肯定是无法满足要求的。所以未来计划不断增加逃逸方法的种类，不仅提供 TCP/IP 层的逃逸方法，还提供应用层的的逃逸方法，包括负载变换，shellcode 变换等，来适应逃逸方法和 IDS 的不断发展。

1.3 对 IDS 提出改善建议

需要注意的是，虽然该系统是逃逸系统，但是该系统从设计之初就不是为了做攻击者，而是作为一个模拟真实的网络逃逸的工具，发现 IDS 的弱点，用来帮助 IDS 的性能。

所以未来计划通过在线下通过尝试各种逃逸方法来逃过 IDS 的检测，一旦发现了逃逸方法可以逃过 IDS 的检测，就可以利用这个 case 去找到能够逃逸的原因，从而更好的帮助改善 IDS 的性能，防止线上被真实的攻击所逃逸。

致谢

时间犹如白驹过隙，转眼间，4 年的大学生活就要画上一个句号了，闭上眼睛，刚入学的情景仿佛就在昨天。马上就要结束在北理工的生涯，去一个另外的地方继续学习，回首自己这四年的经历，自己应该算是非常幸运的。写到这里感触良多，有太多想要感谢的人。

首先必须要感谢我的母校北京理工大学，作为一所一流的大学，你给我们提供了优质的教学资源，同时给了我们一个非常宽松的环境，让我们可以尽可能去做自

己想要做的事情。同时，“德以明理，学以精工”的校训也深深地影响了我，让我脚踏实地，做好自己该做的事情。

然后要感谢我的班主任高春晓老师，辅导员沈丹凤和安冬。是你们给了我一个非常宽松的环境，让我可以自由地去做自己想做的事情。

然后要感谢我的校外导师郑超老师和校内导师高玉金老师，是你们帮助我确定选题，讨论研究方法和实现方案，给了这篇论文很大的帮助。

然后还要感谢实验室的陆秋文师兄，在这篇论文完成过程中，给了我很多的指导与参考建议，没有你的帮助，这篇论文很难完成。然后要感谢实验室的汤琦师兄，在实验过程中的技术问题上提供了很多帮助，帮我解决了很多技术难题。然后还有感谢实验室的张斌师兄，李响师姐，和杨泞构师兄，在论文完成过程中，帮我解决了很多疑惑。

我还要感谢我的室友们，感谢你们的优秀，让我可以跟着你们变得更好，没有你们的帮助和鼓励，我不会保研到中科院，也不会想到去百度实习，总之，感谢你们，希望我们都有一个光明的未来。

最后，我要感谢我的父母，无论我选择做什么，你们都永远站在我背后坚定的支持我，给我提供了坚强的后盾。谢谢你们。

参考文献

. [1] P. C. Lin, Z. X. Li, Y. D. Lin, Y. C. Lai and F.C. Lin, “Profiling and accelerating string matching algorithms in three network content security applications,” IEEE Commun. Surveys Tutorials, vol. 8, issue 2, pp. 24- 37, Second Quarter 2006.

. [2] R. Sommer and V. Paxson, “Outside the Closed World: On Using Machine Learning For Network Intrusion Detection,” In Proc. IEEE Security and Privacy, May 2010.

. [3] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras and B. Stiller, “An Overview of IP Flow-Based Intrusion Detection,” IEEE Commun. Surveys Tutorials, vol. 12, issue 3, pp. 343-356, Third Quarter 2010.

. [4] T. H. Ptacek and T. N. Newsham, “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection,” Technical Report from Secure Networks, Inc., http://insecure.org/stf/secnet ids/secnet ids.html, Jan. 1998.

. [5] M. Handley, V. Paxson, and C. Kreibich, “Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-end Protocol Semantics,” In Proc. USENIX Security Symposium, Aug. 2001.

. [6] R.Lippmann,J.W.Haines,D.J.Fried,J.KorbaandK.Das,“The1999 DARPA off-line intrusion detection evaluation,” Computer Networks, vol. 34, issue 4, pp. 579-595, Oct. 2000.

. [7] H. Debar and B. Morin, “Evaluation of the Diagnostic Capabilities of Commercial Intrusion Detection Systems,” In Proceedings Recent Advances in Intrusion Detection (RAID), Oct. 2002.

. [8] G. Vigna, W. Robertson, and D. Balzarotti, “Testing Network-based Intrusion Detection Signatures Using Mutant Exploits,” In Proc. 11th ACM Conference on Computer and Communications Security (CCS), Oct.2004.

. [9] D. J. Chaboya, R. A. Raines, R. O. Baldwin and B. E. Mullins, “Network Intrusion Detection: Automated and Manual Methods Prone to Attack and Evasion,” IEEE Security & Privacy Magazine, vol. 4, no. 6, pp. 36-43, Nov./Dec. 2006.

. [10] S. Zanero, “Flaws and frauds in the evaluation of IPS technologies,” In Forum of Incident Response and Security Teams, June 2007.

. [11] Metasploit encoding, http://www.derkeiler.com/pdf/Mailing-Lists/securityfocus/pen-test/2006-03/msg00253.pdf.

. [12] R. Bidou, “IPS shortcomings,” In Black Hat Briefings, July 2006.

. [13] D. Roelker, “HTTP IDS evasions revisited,” Technical report, Sourcefire, Sept. 2004.

. [14] T. Detristan, T. Ulenspiegel, Y. Malcom and M. Underduk, “Polymor- phic Shellcode Engine using Spectrum Analysis,” Phrack, 11(61), Aug. 2003.

. [15] O. Kolesnikov and W. Lee, “Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic,” In Proc. USENIX Security Symposium, Aug. 2006.

. [16] P. Bania, “Evading network-level emulation,” Available at piotrbania.com/all/articles/pbania-evading-nemu2009.pdf, June 2009.

. [17] G. Rohrmair and G. Love, “Using CSP to Detect Insertion and Eva- sion Possibilities within the Intrusion Detection Area,” In Proc. BCS Workshop on Formal Aspects of Security, Dec. 2002.

. [18] U. Shankar and V. Paxson. “Active Mapping: Resisting NIDS Evasion without Alerting Traffic,” In Proc. IEEE Symposium on Security and Privacy, May 2003.

. [19] G. Taleck, “Ambiguity Resolution via Passive OS Fingerprinting,” in Proc. International Conference on Recent Advances in Intrusion Detection (RAID), Sept. 2003.

. [20] S. P. Chung, A. K. Mok, “Swarm Attacks against Network-Level Emulation/Analysis,” In Proc. 11th international symposium on Recent Advances in Intrusion Detection (RAID), Sept 2008.

. [21] P. Fogla and W. Lee, “Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques,” In Proc. ACM Conference on Computer and Communications Security (CCS), Oct.–Nov. 2006.

. [22] D. Watson, M. Smart, G. R. Malan and F. Jahanian, “Protocol Scrubbing: Network Security Through Transparent Flow Modification,” IEEE/ACM Trans. Netw., vol. 12, issue 2, pp. 261-273, Apr. 2004.

. [23] H. Dreger, C. Kreibich, V. Paxson and R. Sommer, “Enhancing the accuracy of network-based intrusion detection with host-based context,” In Proc. Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), July 2005.

. [24] A. Pasupulati, J. Coit, K. Levitt and F. Wu, “Buttercup: On Network- based Detection of Polymorphic Buffer Overflow Vulnerabilities,” In Proc. IEEE/IFIP Network Operation and Management Symposium, May 2004.

. [25] U. Payer, P. Teufl and M. Lamberger, “Hybrid Engine for Polymorphic Shellcode Detection,” In Proc. Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), July 2005.

. [26] M. Polychronakis, K. G. Anagnostakis and E. P. Markatos, “Network- level Polymorphic Shellcode Detection using Emulation,” In Proc. Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), July 2006.

. [27] M. Polychronakis, K. Anagnostakis and E. P. Markatos, “Emulation- Based Detection of Non-self-contained Polymorphic Shellcode.” In Proc. 10th International Symposium on Recent Advances in Intrusion Detection (RAID), Aug. 2007.

. [28] K. Borders, A. Prakash and M. Zielinski, “Spector: Automatically Analyzing Shellcode,” In Proc. Annual Computer Security Applications Conference (ACSAC), Dec. 2007.

. [29] C. M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. K. Debray, J. Hartman, “Protecting against Unexpected System Calls,” In Proc. 13th Usenix Security Symposium, Aug. 2005.

. [30] M. Shimamura and K. Kono, “Yataglass: Network-Level Code Emula- tion for Analyzing Memory-Scanning Attacks,” In Proc. Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), July 2009.

.