入侵检测系统
Snort 的不足与改进

潘 军 李祥和

近年来，互联网在国际上得到了长足的发展，因此网络本身的安全性问题也就显得更为重要，而入侵检测则是这两年来在网络安全领域比较热门的技术。Snort是一款非常优秀的基于误用的入侵检测系统。它是主要采用C语言编写的开源软件，在开放源代码软件界非常有名，而且很实用。

一、Snort 的基本原理

从本质上来说Snort就是先对每种攻击提取出其特征值并按照规则语言把它写成检测规则，然后把捕获来的数据包进行解析后再与规则相匹配，若匹配成功则为攻击行为。Snort采用的是基于规则的网络信息搜索机制,对数据包内容进行规则匹配来检测多种不同的入侵行为和探测活动。例如缓冲区溢出，隐藏端口扫描、CGI攻击、SMB探测等等。它之所以采用基于规则的工作方式是因为它的规则检测机制十分简单和灵活，可以迅速地对新的入侵行为做出反应。如它可以在安全漏洞公布不到几个小时的时间内就有相应的Snort规则发布出来用以迅速填补网络中潜在的安全漏洞。

二、Snort 的系统结构

Snort由三个重要的子系统组成:数据包嗅探和解码器子系统、检测引擎子系统、报警与日志子系统。其体系结构流程图如图1。它的工作过程是首先利用Libpcap从网卡上捕获数据包并送交给数据包解码器，由解码器对各种协议栈上的数据包进行解析、预处理，然后把处理结果提交给检测引擎

(上接17页)

(2) 漏洞资料库的组织

由于漏洞模拟系统实际上是分析扫描器发出的探测包中的是否含有探测特征码并返回具有相应响应特征码的数据包。因此，对每一个漏洞，探测特征码和响应特征码是两项必需的描述。 为了便于快速查找，我们把漏洞分类项也加入到描述项中，因此，一个漏洞可以用漏洞名称、服务类型、服务程序、探测特征码、响应特征码来完整地描述。例如，IIS的Unicode目录遍历漏洞可以描述为：

”Unicode目录遍历漏洞“ ”WWW“ ”%“ ”%c1%1c ””200 Ok“

采用数据库技术可以方便地向漏洞资料库中添加新发现的漏洞，使漏洞模拟软件能够不断地更新漏洞资料库，可以更加有效地测试扫描器对安全漏洞的检测能力。

(3) 模拟漏洞编程实现

1) 模拟网络服务漏洞的实现

对每一种网络服务，漏洞模拟软件都提供一个服务代理模块。在主界面上进行适当的配置（如网络服务类型的选择，服务程序和服务端口的选择等）便可启动服务代理线程，使其在相应的端口进行监听。当建立连接后，代理模块分析接收到的数据包，查找数据包中是否存在漏洞探测特征码，如果存在，则返回含有响应特征码的应答包。如果不存在，则按服务协议进行正常应答。

2) 模拟操作系统漏洞的实现

操作系统常见的漏洞主要集中在弱口令、远程共享服务和RPC服务上，这些服务也是与特定的端口相关联（如Windows系统中的135、137、138、139、445等端口)。对操作系统的漏洞模拟比对网络服务漏洞的模拟要复杂一些，因为必须要屏蔽掉操作系统自身对于漏洞探测包的响应。在模拟漏洞实现中可以采用网络数据包截获技术，截获相应端口的数据包，判断是否属于已知漏洞的探测包。如果是漏洞探测包，则模拟漏洞应答，让扫描器认为漏洞存在。反之，则递交给操作系统，让操作系统做出响应。

本文针对目前网络漏洞扫描器测试过程中出现的问题，提出了利用软件来模拟测试环境中的各种漏洞，并阐明了其原理和实现方法。软件模拟漏洞环境可以在两台主机上模拟出两个平台下的各种已知漏洞，利用漏洞模拟软件可以方便、客观、全面地对扫描器的性能进行测试。(作者单位 四川大学信息安全所)

进行规则匹配。Snort采用一个二维链表来把它的检测规则存储在检测引擎中，它每条规则都可以分成逻辑上的两个部分：规则头和规则选项。规则头包括：规则行为、协议、源/目的IP地址、子网掩码以及源/目的端口。规则选项包含报警信息和异常包的信息(特征码,signature),使用这些特征码来决定是否采取规则规定的行为。规则的匹配查找采用递归的方法进行，检测机制只针对当前已建立的规则进行匹配。如果有匹配的数据包则报警与日志子系统就进行响应并记录日志。

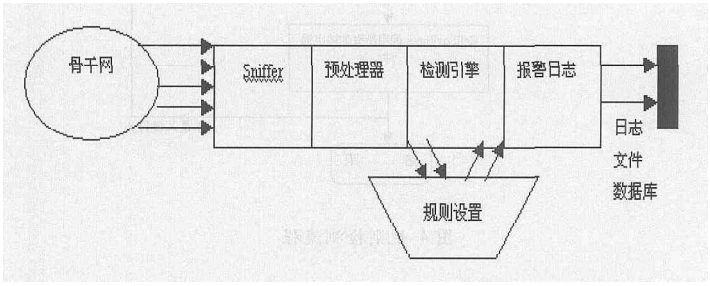

图1 Snort 的体系结构

如果从功能的角度来说Snort分为八个模块,其中包括主控模块、解码模块、规则处理模块、预处理模块、处理插件、输出插件、日志模块、辅助模块。其流程如图2：

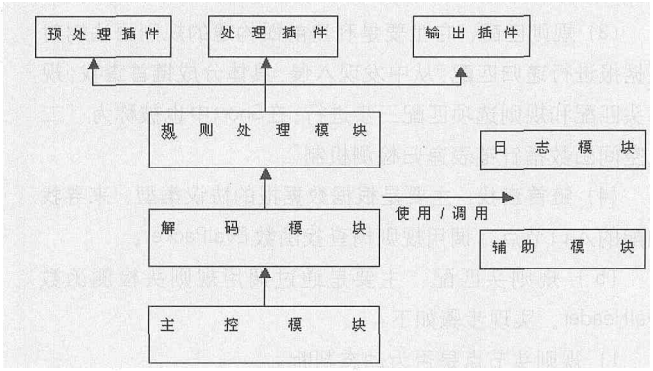

图2 Snort的总体模块图

它的工作过程与前所述一样，只是其中嗅探和解码器子系统的功能相当于主控模块、解码模块和预处理插件三者的功能，检测引擎子系统相当于规则处理模块和处理插件的功能，而报警与日志子系统的功能就是日志插件和输出插件的功能。另外在系统运行过程中还使用了一些辅助模块。

三、Snort 的规则解析流程

Snort之所以能完成入侵检测的功能是因为它的规则文件就是它的攻击知识库，在它的规则库中每条规则都含有一种攻击标识，Snort 就是通过它的不同规则来识别不同的进攻。Snort 使用一种简单的，轻量级的规则描述语言。这种描述语言易于扩展，功能也比较强大。

1、规则的基础

下面是一条简单的规则：

alert tcp any any -> 192.168.1.0/24 111(content: "|00 01 86 a5|";msg: "mountd access";)

Snort的规则在逻辑上分为两部分:规则头(Rule Header)和规则选项(Rule Option)。从开头到最左边括号属于规则头部分,它包括:规则行为(rule 's action)、协议(protocol)、源/目的IP地址、子网掩码以及源/目的端口。括号中的部分是规则选项，它包含报警信息和异常包的信息(特征码,signature),使用这些特征码来决定是否采取规则规定的行动。

2、规则的组成

(1)规则头：包含一个报文关键的地址信息、协议信息以及当报文符合规则时各元素应该采取的行动。每条规则的第一项就是规则行为(rule action)。规则行为告诉Snort当发现匹配的数据包时，应该如何处理。

(2)协议字段：每个规则的第二个域是协议字段。当前Snort支持对四种协议的分析：IP、TCP、UDP和ICMP,在将来它还会支持更多的协议，如ARP、IGRP、RIP等。

(3)地址和端口信息：规则头下面的部分就是IP地址和端口信息。关键词any可以用来定义任意的IP地址。Snort不支持对主机名的解析。所以地址只能使用数字/CIDR的形式。/24表示一个C类网络；/16表示一个B类网络；而/32表示一台特定的主机地址。例如：192.168.1.0/24 表示从192.168.1.1 到192.168.1.255 的地址。

(4)规则选项:对规则选项的分析构成了Snort的检测引擎的核心，既易用又非常灵活强大。首先其灵活性是指可以根据不同的行为制定相应的检测内容,其强大性是指不仅检测性具有一定的广度和深度并且定义了检测到时该做什么。它的所有选项使用分号"；"分隔，选项的关键字和它的值之间使用冒号分隔。例如这三个关键字就是作为检测到以后的回应：

msg —在报警和日志中打印一个消息

logto—把包记录到用户指定的文件中而不是记录到标准输出。

resp —主动反应（切断连接等）

3、Snort的规则解析流程

Snort的规则解析流程很简单：它首先读取规则文件,紧

接着依次读取每一条规则，然后对其进行解析，并用相应的规则语法表示。在内存中对队进行组织，建立规则语法树。图3描述了Snort的规则在内存中的逻辑关系。

从图中可以看出，所有的规则按照规则头排成主链，然后根据规则选项把规则插入到这个链中，构成一颗规则树，这样每一个选项节点就对应一条规则。规则头节点主要记录了规则头信息，包括源IP、端口、目标IP/端口，并有指针指向下一个规则头节点、附属于它的选项列表和规则头列表结构。规则选项节点存放所有有着规则选项的信息和处理插件的处理函数列表(option_func)，分别指向规则头节点的指针和关联选项节点的指针。

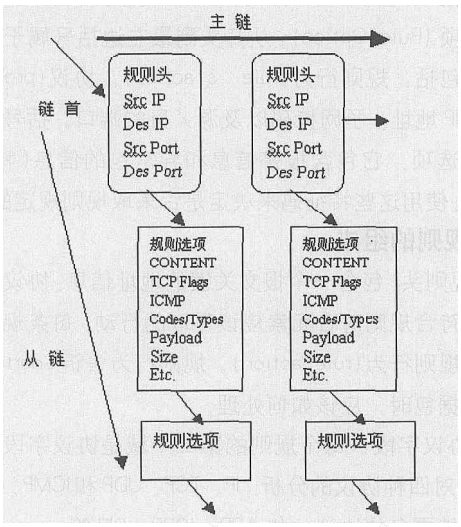

图3 Snort 规则内存表示逻辑图

Snort规则是根据源IP、目标IP、源端口范围和目标端口范围等属性分类组织成为规则集的。当一个数据包被检测时，Snort从左向右检测每个规则集的上述四个参数以决定是检测该规则集还是转移到下一个规则集。如果数据包与规则集的四个参数相匹配，该规则集中的规则依次得到检测，并且每条规则中的其余的参数也按顺序检测。当该规则集中的所有规则被检测完以后，如果不匹配则接着搜索下一条规则集中的四个参数，检测过程重新开始。

四、Snort 的规则检测流程

我们以单个数据包的检测流程进行详细的分析：首先对收集到的数据报进行解码，然后调用预处理函数对解码后的报文进行预处理，再利用规则树对数据进行匹配。在规则树匹配的过程中，Snort要从上到下依次对规则树进行判断，从链首、链表到规则头节点，一直到规则选项节点。其检测流程如图4：

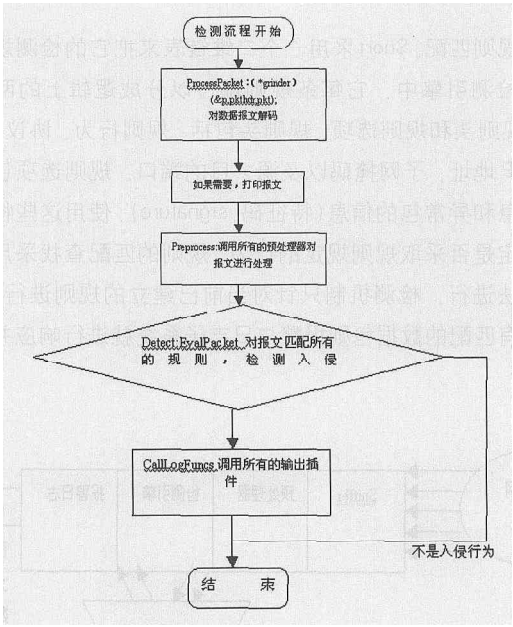

图4 规则检测流程

具体检测步骤如下：

(1) 数据报解码：主要是调用decode.c中的各种解码函数来对报文进行分析的过程，这些报文解码函数包括DecodeEthPkt、DecodeFDDIPkt、DecodeICMP、DecodeTCP等

(2) 预处理：则主要是调用Preprocess预处理函数列表中的函数对解码后的数据报进行匹配前的预处理，这些函数功能包括数据分片重组、流重组、代码转换等等。

(3) 规则匹配：它主要是利用前面构建的规则语法树对数据报进行递归匹配，从中发现入侵，具体分成链首查找、规则头匹配和规则选项匹配三步进行。在Snort中也被称为“二维空间函数指针链表递归检测机制”。

(4) 链首查找：主要是根据数据报的协议类型，来寻找匹配的入口节点，调用规则树查找函数EvalPacket。

(5) 规则头匹配：主要是通过调用规则头检测函数EvalHeader。实现步骤如下：

1) 规则头节点是否为动态判断；

2) 调用规则头检测函数EvalHeader进行IP地址、端口、数据流方向等的匹配；

3) 如果匹配，把可疑数据报发给EvalOpts规则选项匹配函数，进行进一步检测；

4) 如果检测成功，调用响应函数TriggerResponses进行响应操作；

5) 再根据需要，调用SetTags函数设置tag高级日志操作标志位；

6) 对确认为入侵行为的数据报调用相应的处理函数(如AlertAction、DynamicAction或LogAction等)进行忽略、告

警、日志和激活等操作;

（6）规则选项匹配：主要是调用规则选项检测函数EvalOpts。实现步骤如下：

1）规则选项节点是否为动态判断;

2）调用规则选项匹配函数进行规则选项部分的匹配;

3）如果匹配成功，就把检测结果返回给规则头检测。

（7）TAG列表匹配：调用CheckTagList进行TAG相关的检测，以记录必要的信息。

五、Snort规则的测试

1、规则的压力测试

我们采用Snort/stick这种IDS的压力测试工具，它的测试原理是先读取Snort的规则集，然后按规则的选项描述生成相应的数据包发送给IDS，这样短时间内Snort会认为出现了大量的“攻击”，随着数量的增多可能会使IDS处理能力溢出，造成崩溃或失去工作能力。

测试如下：

首先采用详细报警，不记录数据包的模式启动Snort进行测试。

[root@redhat72 snort]# snort -D -N -A full 用Snot进行攻击，测试中snort的CPU平均占用率在10%-20%之间，测试几秒钟后，日志文件这时已产生了大量的报警信息

[root@redhat72 snort]# ls -l

total 12302

-rw------- 1 root root 11600253 Apr 10 10:56 alert

当我们加上记录数据包和解码模式启动Snort后，这时会看到CPU的平均占有率提高了两倍左右。

2、状态相关的TCP攻击测试

Snort新增的Stream4预处理器，因为它可以跟踪TCP连接的建立情况，所以可以忽略掉那些”无状态“的数据包。

用一般方式启动snort。

[root@redhat72 bin]# ./Snort -Dd -N -A full 用snot攻击几秒钟后，看看产生的日志文件大小，可以看到产生了相当多的报警信息。

[root@redhat72 snort]# ls -l

total 573

-rw------- 1 root root 579536 Apr 10 11:33 alert

现在我们加入参数-z est（它能使Snort对违反状态的包不进行检测），启动Snort。

[root@redhat72 bin]# ./snort -D -N -A full -z est 。

用Snort攻击相同的时间，查看日志文件的大小：

[root@redhat72 snort]# ls -l

total 241

-rw------- 1 root root 206596 Apr 10 11:15 alert

可以看到报警信息少了许多，通过查看alert文件里的具体报警信息，发现需要建立真正TCP连接的假攻击都已经被忽略了。

3、测试问题的总结

（1）如果我们强行删除alert报警文件，则Snort不会重建它，因此就失去了记录报警信息的功能。

（2）进行测试中，发现Snort在处理很多源IP与目标IP相同的畸形包（无三次握手过程）时，CPU占用率会达到很高。

（3）Snort规则的通配符匹配选项很多时候并不能在正常状态下工作。

（4）由于Snort完全由规则驱动，它所做的只是对到网络接口的数据包做生硬的规则匹配，虽然通过预处理器比如stream4引入一些底层数据包相关的状态检测，但还远远不够，因此基本上不能检测一些高层协议状态相关的攻击。Snort虽然对某些应用广泛的应用层协议如HTTP、TELNET等提供了解码插件，能对特定端口上的数据进行解码分析，但并没有达到对高层协议的状态进行跟踪的程度。

（5）Snort的规则数据库还极不完善，每条规则的相关说明文档非常不完整，某些规则的创建并没有经过严格的测试，会带来不少可以避免的漏报和误报。

本人认为Snort应提供一种对日志和报警等重要组件的保护和自动生成机制以确保自身的安全性。还应建立个高层协议状态处理器用于跟踪和反馈高层协议状态信息，根据跟踪的结果做出响应，也可采用异常检测的方式把一些其它状态信息作为检测条件以增强其检测能力。

通过实际的应用我们发现由于Snort定位在轻量级的系统，所以它不可避免的会产生一些不足。

Snort虽然能很有效地对单包的特征进行匹配，但对于只与状态相关的攻击它基本上没有能力描述特性，所以也很难用来探测一些与状态相关的攻击。这就与Snort在规则集上采用的检测方式－误用检测有关，它不具有异常检测功能，因此仅对检测已知的攻击有效，对于未知的其漏报率和误报率会很高。并且规则的自适应，自学习能力不高。

我们认为Snort已经具备了NIDS的基本功能，由于它本身定位在一个轻量级的入侵检测工具，所以与商业的入侵检测工具比起来规则语言略显简陋，但相信通过不同程序员对它的维护和升级会变得更加完善。（作者单位 解放军信息工程大学信息工程学院）