预处理插件在 IPv6 入侵检测系统中的应用①

张 宏, 龙 春, 葛敬国, 李 俊

(中国科学院 计算机网络信息中心, 北京 100190)

摘 要: Snort 是一个功能强大的轻量级 NIDS, 它能够检测出各种不同的攻击方式, 并能对攻击进行实时告警. 针对 Snort 没有提供对 IPv6 地址前缀攻击检测支持的问题, 提出利用预处理插件检测该类攻击的解决方案, 给出了插件的检测流程. 实验结果表明, 该插件对地址前缀欺骗攻击具有较高的检测率, 是一种有效的入侵检测系统插件.

关键词: 预处理插件; IPv6 网络; 地址前缀; Snort 入侵检测

Application of Preprocessor Plug-in to IPv6 Intrusion Detection System

ZHANG Hong, LONG Chun, GE Jin-Guo, LI Jun

(Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Snort is one of a light wight, powerful NIDS. It can detect many different kinds of attack behaviors, and give real-time alerts. Because Snort does not support the detection of IPv6 address prefix spoofing, this paper supplies a solution to implement the intrusion detection preprocessor plug-in and provides the detection process. The experimental result proves that the preprocessor has a higher detection ratio to the spoof of IPv6 address prefix, and it is an effective plug-in on the intrusion detection system.

Key words: preprocessor plug-in; IPv6 network; address prefix; Snort intrusion detection

1 引言

随着 IPv6 协议研究的深入, 在 IPv4 向 IPv6 过渡阶段的网络安全领域中出现了许多新课题. 基于 ICMPv6 的邻居发现协议 NDP(Neighbor Discovery Protocol)解决了同一条链路上的节点间的交互问题, 取代了 IPv4 中的 ARP 协议、ICMP 路由发现和 ICMP 重定向功能, 并具备获取链路 MTU、跳数限制等网络参数的机制. IPv6 邻居节点的发现过程就是用一系列的报文和步骤来确定邻节点之间的关系, 进行网络配置的过程. 由于在本地链路上节点的加入不需要通过任何认证, 从而导致了诸多安全问题. 如链路内的恶意主机可以通过发送包含错误子网前缀的 RA(Router Advertisement)报文, 使被攻击主机使用错误的前缀信息配置本机接口的 IPv6 地址, 导致该主机无法进行正常的网络通信. 为应对 NDP 协议的安全威胁, RFC 3971 提出了 SEND 协议[1], 不过该机制还没有广泛应用.

Snort 是一个符合 CPL 规范的网络入侵检测工具, 也是目前使用最为广泛的入侵检测系统[2], 它通过对数据包的协议解码、预处理, 把数据送入检测引擎, 检测引擎根据规则库中的规则对每一个数据包进行内容查找及匹配, 从而判断是否发生了入侵行为.

由于 Snort 引入了插件机制, 所以很容易对 Snort 进行功能的修改和增加. Snort 包含 3 种插件: 预处理插件、检测插件和输出插件. 其中, 预处理插件灵活性最强, 能在 Snort 的规则匹配核心之外提供众多功能, 主要有包重组、协议解码和异常检测[3]. 由于 Snort 采用规则匹配的方式检查网络中的异常数据包, 有些攻击行为是无法用规则描述的, 因此为了让 Snort 具有更好的攻击识别能力, 往往需要用到预处理插件. 与此

① 收稿时间:2011-12-23;收到修改稿时间:2012-02-06

同时，系统现有的预处理插件仅支持对 IPv4 协议数据包的处理，因此研究预处理插件在 IPv6 入侵检测系统中的应用就显得尤为迫切. 本文通过挖掘伪造 IPv6 地址前缀攻击实例的特征，以 Snort 2.9.0.5 为基础，设计并实现一个检测 IPv6 地址前缀欺骗攻击的预处理插件.

2 IPv6编程接口简介

在 IPv4 的地址结构中，地址包含在 in_addr 中. 而 IPv6 定义了一个更大的结构 in6_addr 来保存更长的 IPv6 地址.

```
struct in6_addr {
   union {
            uint8_t u6_addr8[16];
            uint16_t u6_addr16[8];
            uint32_t u6_addr32[4];
       } in6_u; };
```

除了更长的地址，在 IPv6 的地址结构中还要包含其它的用于 IPv6 的成员，如地址簇、端口和地址信息、通信流类别、流标签、范围 ID 等，结构 sockaddr_in6 如下:

```
struct sockaddr_in6 {
   sa_family_t     sin6_family;
   in_port_t       sin6_port;
   uint32_t        sin6_flowinfo;
   struct in6_addr sin6_addr;
   uint32_t        sin6_scope_id;
};
```

IPv4 通过 inet_ntoa 和 inet_aton 函数完成文本表示的十进制点分地址和 32 位网络字节序地址之间的转换. IPv6 通过 inet_pton 和 inet_ntop 函数完成文本 IPv6 地址和 128 位网络字节序地址之间的转换. 例如，inet_pton(AF_INET6,"2001:cc0:2fff:1:be30:5bff:fea7:eb51", &glob_in6) 用于把字符表示的 IPv6 地址“2001:cc0:2fff:1:be30:5bff:fea7:eb51”转换成 in6_addr 结构，并复制到 glob_in6 变量中.

在域名解析方面，IPv4 使用 gethostbyname 完成主机名或域名到 IP 地址的解析，而 IPv6 使用独立于协议之上的函数 getaddrinfo 和 getnameinfo. IPv4 与 IPv6 编程接口之间的差异如表 1 所示.

表 1 IPv4 与 IPv6 编程接口间的差异

名 称	IPv4	IPv6	说 明
地址结构	struct in_addr	struct in6_addr	存储 IP 地址的结构
	struct sockaddr	struct sockaddr_in6	存储 IP 地址的结构
地址转换函数	inet_aton()	inet_pton()	字符串 IP 地址转换为网络字节序地址
	inet_ntoa()	inet_ntop()	网络字节序地址转换为字符串 IP 地址
域名解析函数	gethostbyname()	getaddrinfo()	主机名/域名解析为 IP 地址
		getnameinfo()	IP 地址解析为主机名/域名

3 预处理插件简介

预处理器是作为插件在 Snort 系统中使用的. 预处理插件在检测引擎工作之前，报文解码之后运行，检查并修改数据包. 预处理插件的基本设计思想是: 在系统中建立一条预处理函数链表，根据系统配置文件里配置的预处理服务，将所需的处理函数加入该链表中. 在进行真正的基于规则的检测前，要先沿该链对数据包信息进行处理. 各个预处理函数所需的参数 Packet *p 指向由协议解析引擎解析后的网络数据包信息，参数 void *context 指向与当前预处理插件相关的上下文信息，其检测流程如图 1 所示.

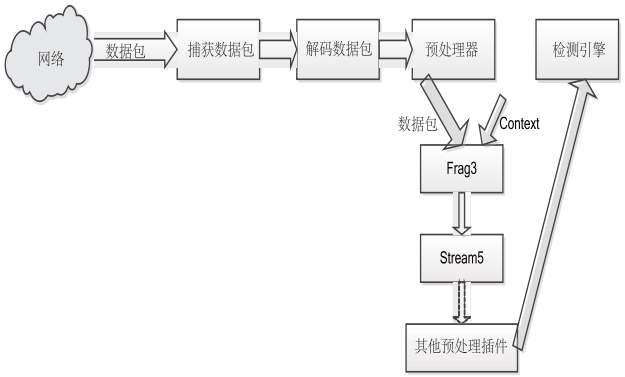

图 1 预处理引擎处理流程

按照实现功能的不同，预处理插件可分为三种：包重组预处理插件、协议分析和规范化预处理插件、异常检测预处理插件. 按照调用时机的不同，可分为静态插件和动态插件. 根据上述分类方法，本文欲实现的IPv6地址前缀欺骗攻击的预处理插件分别属于异常检测预处理插件和静态插件.

一个预处理插件通常由三个框架函数组成[2]：插件安装函数、插件初始化函数、插件的功能执行函数，如表 2 所示，其中 XXX 表示某给定的插件名.

表 2　预处理插件框架函数表

名 称	函数名	何时调用	功 能
插件安装函数	SetupXXX()	注册时调用	注册插件的初始化函数等
插件初始化函数	XXXInit()	解释规则文件时调用	完成插件的初始化，并注册功能执行函数等
插件的功能执行函数	XXXXX()	检测流程中调用	在检测过程中完成预插件的功能

4　检测地址前缀欺骗攻击的预处理插件

4.1 前缀欺骗攻击

在无状态自动配置过程中，节点利用路由器前缀与自己的接口地址生成全局的IPv6地址，如图 2 所示.

图 2　IPv6 地址

接口地址从MAC地址映射而来，首先将MAC地址映射到EUI-64地址，然后对U/L位求反. 图 3 是接口地址的转换的过程.

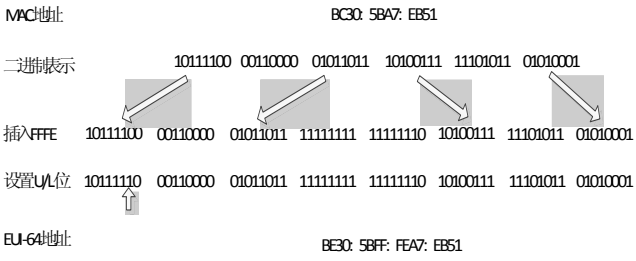

图 3　接口地址的生成

获得路由器前缀的方法有两个[4]，一是节点主动发送 ICMPv6 Router Solicitation 消息，等待 Router Advertisement 消息；二是监听链路上周期广播的 Router Advertisement 消息. 由于这种请求/应答或者监听的方式缺乏一种认证机制，因此很容易被攻击者利用.

节点在与目的节点通信时，节点首先查询本地缓存，查看 Destination Cache 是否有到目的地址的记录. 当 Destination Cache 中没有目的节点的记录，节点则进行 Prefix List 匹配，即根据目的 IP 的前缀匹配情况决定目的节点是否在本链路内，从而确定目的 IP 的下一跳转发 IP 地址. 因此，若攻击节点在链路内发布非本地链路的地址前缀，则当本链路内的节点第一次与 IP 地址以此前缀开头的节点通信时，将认为该 IP 地址属于本地链路，从而直接进行邻居发现，而不是通过默认路由器进行转发. 这样，攻击节点只需在本地链路内冒充 IP 地址以此前缀开头的节点则能实施欺骗，成为“中间人”[5].

例如，假设攻击节点 B 欲截取 A(其中 IPA=2002:9fe2:3acc:b:be30:5bff:fea7:eb51)节点与外网 3001:cc0:2fff:1::/64 内节点 X(IPX=3001:cc0:2fff:1::6)的通信，攻击节点可采取图 4 所示的攻击方法，先在本地链路发送虚假 RA，发布 3001:cc0:2fff:1::/64 为本链路地址前缀的信息，在链路内的节点 A 会根据 3001:cc0:2fff:1::/64 产生基于此前缀的 IP 地址 3001:cc0:2fff:1:be30:5bff:fea7:eb51. 当节点 A 通过 DNS 查询与 X 建立连接时，发现节点 X 和自身具有相同地址前缀，因而认为节点 X 处于本地链路. 因此，节点 A 将发起邻居请求 NS 查询，此刻攻击节点 B 通过回应 NS 就能伪装为节点 X 与 A 通信.

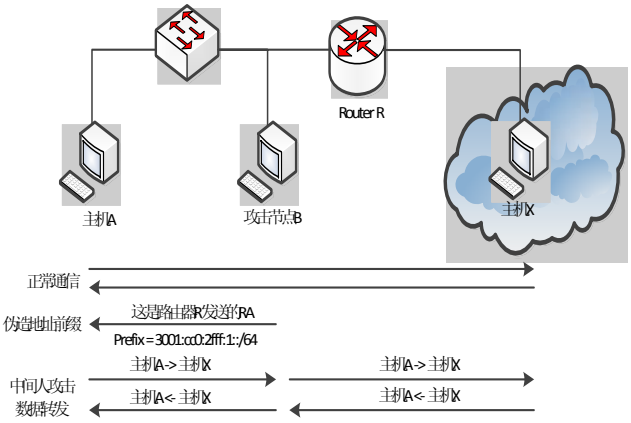

图 4　基于伪造地址前缀的中间人攻击

另外, 攻击节点也可以发送包含错误子网前缀的 RA 报文, 被攻击节点使用错误的前缀信息配置本机接口的 IPv6 地址, 导致该节点无法进行正常的网络通信, 实现拒绝服务攻击.

4.2 检测分析

根据前缀欺骗攻击的原理可知, 其攻击特征是攻击节点假冒路由器给本地链路的所有节点(ff02::1)发送包含虚假的地址前缀的组播报文. 一般的检测思路是对捕获的数据报进行特征分析, 如果特征匹配成功, 则检测到该攻击. 通过对基于地址前缀欺骗的攻击原理分析, 我们可使用组播报文中 MAC 地址、Prefix 信息等字段作为特性进行攻击识别. 但在攻击者修改源代码并使用其他 MAC 地址或 Prefix 信息后, Snort 就无法检测新的攻击, 从而导致漏报率的升高.

既然无法使用被动方式对攻击进行检测, 则监控主机需要主动进行探询来检查网络中是否存在前缀攻击. 检测的思路是: 监测主机以其链路地址为源, 主动向链路中所有的路由器(ff02::2)发送一个路由请求报文, 局域网中正常的主机不对该报文进行应答, 只有链路内的路由器和攻击主机会产生应答. 如果捕获到的路由应答报文中的 Prefix 信息与链路内的主机不相同, 则证明网络中攻击主机的存在. 如果捕获到的 RA 应答报文中的 Prefix 信息与链路内的主机相同, 则证明局域网中不存在前缀攻击.

检测的关键在于对 prefix 信息的选择, 且保证进行前缀欺骗攻击检测的 IP 地址没有被局域网的正常主机使用. 满足要求的一个选择就是监控主机自身的地址前缀信息及 IPv6 地址. 总的来说, 地址前缀攻击的检测思路为: 选择监控主机作为“诱饵”, 诱使攻击出现.

4.3 预处理的检测流程

prefixspoof 预处理插件采用 effectivetime 和 sleeptime 参数进行处理, effectivetime 用于控制报警消息的有效时间, sleeptime 用于控制发送诱饵数据的时间间隔. 实现的检测流程如图 5 所示: 首先进行初始化工作, 将标志 flag 置为 NONE, 并生成用于存储监控主机的全局 IPv6 地址前缀的 arrays 数组以及用于存放时间的变量 tm.

程序分为主线程和分线程分别操作. 分线程控制诱饵数据包的发送. 当 flag=NONE 时, 表明预处理器不清楚网络中是否存在地址欺骗攻击, 则睡眠 sleeptime 一段时间, 然后发送诱饵数据报, 再进行条件判断. 当 flag=DONE 时, 说明预处理器已探测网中存在地址前缀欺骗攻击, 分线程获取当前时间, 并与发现攻击的时间 tm 进行比较, 如果两者的时间差超过预定的有效时间 effectivetime, 说明之前的报警已经失效, 需要重新检测, 将 flag 置为 NONE, 并重新发送诱饵数据报.

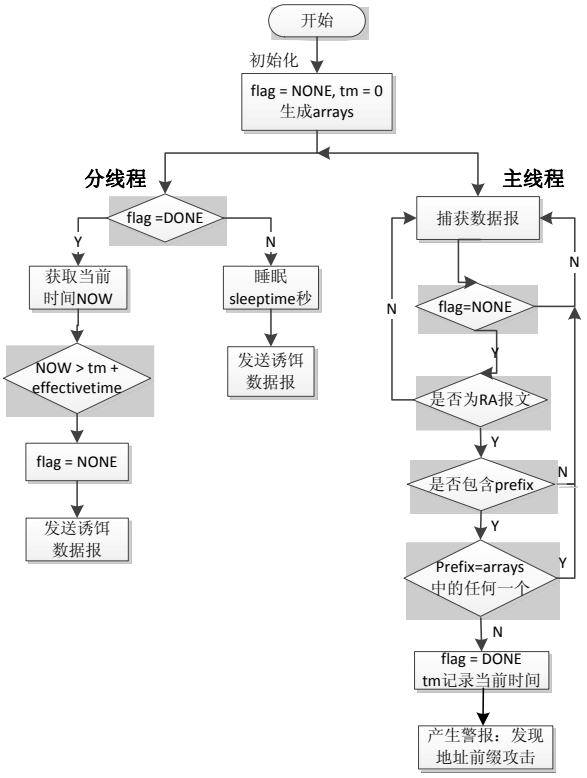

图 5 Prefixspoof 攻击检测流程

主线程进行数据报的检测报警工作. 当 Snort 捕获到数据报后首先测试 flag 标志, 如果值为 NONE 则检查数据报, 如果 RA 报文中包含地址前缀信息 prefix, 则与存储监控主机全局地址前缀信息的 arrays 数组比较, 只要 prefix 与 arrays 中的任一地址前缀相同, 说明该报文为正常的 RA 报文. 如果 prefix 与 arrays 中的任一前缀都不相同, 则说明发生了前缀攻击, 将 flag 置为 DONE, 记录当前时间到 tm 并发送警报信息.

4.4 系统实现

prefixspoof 插件使用预处理插件链表来进行操作, 而预处理插件链表用到了四个重要的数据结构: PreprocConfig、PreprocConfigFuncNode、PreprocEval FuncNode 和 PreprocSignalFuncNode, 如图 6 所示.

PreprocConfig 封装了预处理插件的关键字、配置文件名、预处理插件在配置文件中所处的位置、配置标记、配置参数等信息. PreprocConfigFuncNode 封装了预处理插件的名称和对应的初始化函数指针、重启函数指针等信息, PreprocEvalFuncNode 封装了预处理插件的功能执行函数指针、预处理插件 ID 和优先级等. PreprocSignalFuncNode 封装了预处理插件 ID 和对应的清除退出函数指针等.

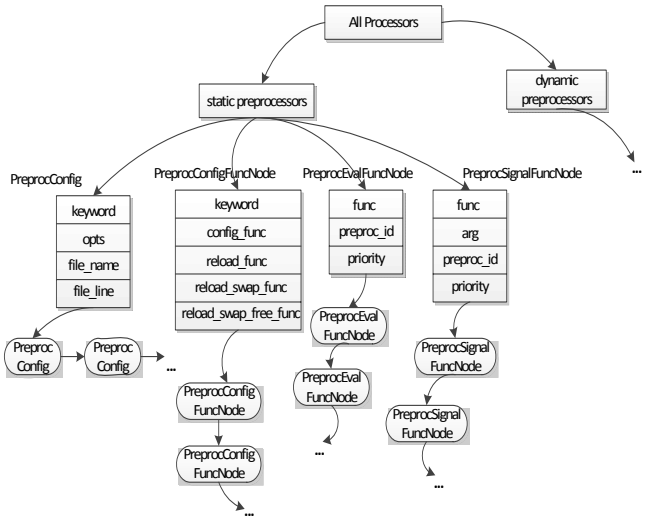

图 6　预处理插件链表结构

Prefixspoof 预处理插件的工作主要分为三步: 注册->初始化->调用.

(1) 注册

主函数调用 RegisterPreprocessors()函数进行预处理插件的注册, 依次调用 SetupXXX 函数实现对每一个预处理插件的注册. 注册的基本思想是将插件与初始化函数、重启函数、重启交换函数和重启交换释放函数进行绑定.

以 Prefixspoof 插件为例, SetupPrefixspoof()函数通过调用 RegisterProcessor 函数来实现注册:

RegisterPreprocessor("prefixspoof", PrefixspoofInit, PrefixspoofReload,PrefixspoofReloadSwap, PrefixspoofReloadSwapFree);

(2) 初始化

注册后, 需要从 snort.conf 配置文件中读取预处理插件的配置参数. 即从配置文件中把用户配置预处理插件对应的信息读取出来, 分析参数, 将配置参数交由对应的预处理插件的初始化函数来完成. 例如, 在 snort.conf 中配置 prefixspoof 预处理插件“preprocessor prefixspoof: 5 2”, 表示 prefixspoof 插件的 effectivetime 为 5 秒, sleeptime 为 2 秒. ConfigurePreprocessors 函数进一步拆分分析后, 提取出预处理插件的名称 prefixspoof. 然后遍历预处理插件初始化时建立的链表, 找到与 prefixspoof 相关的配置节点. 由于在注册插件时, 节点中已经封装了相应的初始化函数, 那么直接调用

node->config_func(config->opts)

将配置参数“5 2”传递给对应的初始化函数 PrefixspoofInit 进行插件的初始化操作.

初始化函数 PrefixspoofInit 完成如下工作:

①拆解分析配置参数, 并填充相应的数据结构. 插件使用的数据结构为:

```
typedef struct _PsContext
{
    int flag;   /*whether send alert msg */
    time_t tm;      /* time of send alert msg */
     struct in6_addr prefix;     /*prefix info of caputured pkt */
      uint8_t prefix_len;        /*prefix length of caputured pkt */
    struct in6_addr arrays[8]; /* get prefix info from sensor*/
    int num_array;   /*number of prefix*/
} PsContext;
```

② 将该预处理插件的实际功能执行函数 PrefixspoofCheck 填充至预处理评估函数链表 PreprocEvalFuncNode 中.

③ 将该预处理插件的退出清除函数 PrefixspoofCleanExit 填充至预处理信号函数链表 PreprocSignalFuncNode 中.

(3) 调用

当所有的预处理插件初始化工作完成后, 系统构建完成一条由这些插件的实际功能执行函数组成的链表, 待由捕获到数据包后对数据包进行逐一地调用, 对数据包进行预处理.

在数据包进入检测引擎之前, Preprocess()函数将逐一调用所有初始化过的预处理插件的实际功能执行函数.

PreprocEvalFuncNode*idx=policy->preproc_eval_f

```
uncs;
    for(; (idx!=NULL)&& ....; idx = idx->next)
    {
      ... ...
      idx->func(p, idx->context);
    }
```

Prefixspoof 插件的功能执行函数 Prefixspoof Check()实现如 5.3 节所述的检测流程.

5 实验测试

5.1 测试环境

测试环境包括一台路由器、一台交换机及其一台攻击主机、一台监控主机和主机 A. 其中, 在交换机上配置了端口映射, 使得监控主机可以监测全网流量, 并在监控主机上部署 Snort 及 Prefixspoof 插件. 主机 A 安装 windows XP, 攻击主机和监控主机安装 Fedora 11, 如图 7 所示.

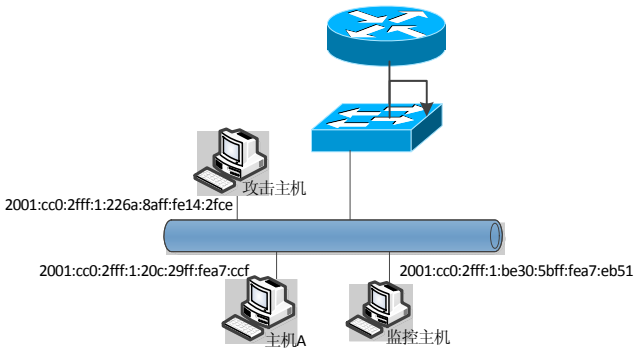

图 7 测试环境

5.2 测试分析

为了测试 Prefixspoof 预处理插件防御 IPv6 地址前缀欺骗攻击的效果, 在攻击主机上部署 http://www.thc.org 的 THC-IPv6 Attack Toolkit 套件中的 fake_router6 工具对主机 A 进行 IPv6 地址前缀攻击测试, 结果表明系统能比较有效地检测出这些攻击. 表 3 为入侵测试结果.

表 3 测试结果

攻击次数	检测到的攻击次数	成功率	漏报率
260 次	256 次	98.46%	1.54%

6 结语

本文在研究 IPv6 地址前缀攻击的基础上, 分析 Snort 预处理插件的工作机理, 实现了一个识别该类攻击的预处理插件, 扩充了 Snort 的攻击识别能力. 实验证明了该插件具有较高识别率.

预处理插件虽然功能强大, 但与检测引擎相比更耗费资源. 预处理器越多, Snort 性能受到的影响越大, 并且根据预处理插件功能性质的不同, 造成性能下降的程度也不同. 在未来的工作中, 将研究预处理插件的性能优化问题, 一定程度上避免因使用不必要的预处理器影响 Snort 的性能.

参考文献

1 Arkko J, Kempf J, et al. RFC 3971-SEcure Neighbor Discovery (SEND).(2005-03). http://www.ietf.org/rfc/rfc3971.txt

2 韩东海,王超,李群.入侵检测系统及实例剖析.北京:清华大学出版社,2002.

3 汪兴东,周明天.基于 BP 神经网络的智能入侵检测系统.成都信息工程学院学报,2005,21(1):1–4.

4 杜敏政. IPv6 下网络攻击的研究与实现.北京:清华大学, 2008.

5 张程亮,吕宇鹏. IPv6 NDP 协议的中间人攻击研究.2010 年全国通信安全学术会议论文集.2010.120–122.