需求装载代码协议的安全缺陷分析*

夏正友[1+], 蒋嶷川[2], 钟亦平[3], 张世永[3]

[1](南京航空航天大学 计算机科学系,江苏 南京 210016)

[2](香港浸会大学计算机科学系,香港)

[3](复旦大学 计算信息技术系,上海 200433)

Analysis for Security Flaw in Demand Loading Code Protocol

XIA Zheng-You[1+], JIANG Yi-Chuan[2], ZHONG Yi-Ping[3], ZHANG Shi-Yong[3]

[1](Department of Computer Science, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

[2](Department of Computer Science, Hongkong Baptist University, Hong Kong)

[3](Department of Information & Technology, Fudan University, Shanghai 200433, China)

+ Corresponding author: E-mail: zhengyou_xia@yahoo.com, http://www.nuaa.edu.cn

Received 2003-08-07; Accepted 2004-05-14

Xia ZY, Jiang YC, Zhong YP, Zhang SY. Analysis for security flaw in demand loading code protocol. *Journal of Software*, 2005,16(6):1175−1181. DOI: 10.1360/jos161175

Abstract: In this paper, the authors use SPI calculus to analyze the demand loading code protocol of active network. The security flaw of being replay attack in this protocol is found. Model of active network is different from the tradition network. Since the model of active network is compute-storage-forward, and the model of tradition network is storage-forward. The replay attack would make the unexpected result for active network and letdown the performance and efficiency of active node. In order to prevent the replay attack, the authors amend the protocol and enhance the capacity for preventing replay attack.

Key words: active network; demand loading code protocol; SPI calculus; replay attack

摘 要: 使用 SPI 演算对主动网络的需求装载代码协议进行分析,发现其存在被重放攻击的安全漏洞.由于主动网络是计算-存储-转发模型,不同于传统网络的存储-转发模型,所以这种被重放攻击的安全缺陷将对主动节点产生难以预测的后果,并减低其性能和效率.为了消除被重放攻击的危险,修改了原有需求装载代码协议,并增加了其阻止重放攻击的能力.

关键词: 主动网络;需求装载代码协议;SPI 演算;重放攻击

中图法分类号: TP309 **文献标识码**: A

主动网络[1−3]是 1996 年 DARPA(defense advanced research projects agency)开发组在关于网络系统未来发展方向的讨论会上提出来的.由于今天的网络体系结构面临着几个问题:(1) 新的网络技术和标准很难集成到

* **作者简介**: **夏正友**(1974−),男,安徽巢湖人,博士,讲师,主要研究领域为主动网络,移动 Agent,网络安全;**蒋嶷川**(1975−),男,博士后研究员,主要研究领域为移动 Agent,网络安全;**钟亦平**(1954−),女,副教授,主要研究领域为网络安全;**张世永**(1951−),男,教授,博士生导师,主要研究领域为网络安全.

当前正在使用的网络设备中去,因为现有的网络结构是封闭式系统,网络编程环境仅限于端系统内的高层网络应用软件中,网络的中间节点是不可编程的,因此造成网络技术更新速度缓慢,不能适应新的技术和标准的应用;(2) 由于在现有网络体系中几个协议层的冗余操作造成网络性能不佳,很难将新的应用服务整合到现在的网络中.因为网络设备受现有网络体系限制不能动态更新,所以很难使新的服务应用得到支持;(3) 在现有的网络体系结构下,投资者为了满足市场的需求,不得不为每增加一种新的服务应用而去重新购买新的设备,所以传统的网络结构体系不能保护投资者利益.鉴于以上诸多问题和缺陷,人们开始重新审视现有的网络体系结构,并提出了新的网络体系结构.在这种情况下,出现了从一维网络模型向二维网络模型的转变,即从传统报文的存储和转发模型,到报文的存储、计算和转发模型[3].主动网络的基本思想是:它不像传统网络那样被动地把比特流从一个节点传输到另一个节点,主动网络的报文含有能在中间节点执行的程序代码,报文中所含的代码能够扩展、更改网络设备并能够在其网络设备上执行.

经过几年的主动网络概念和基本原理的研究,比如在主动网络节点抽象模型方面的研究[4],主动网络的执行环境 EE(执行环境,目前已经研究出的有 ANTS[5],Smart packet[6]和 PAN[7]等),主动网络操作系统 OS 方面的研究[8],主动网络安全模型和安全特性方面的研究[9,10~14]都取得了很多的成果.同时,在主动网络协议方面的研究有 ANEP[15~17]等.但是仍然有许多问题还没有解决或需要完善,如在 ANTS[5]和 PAN[7]中的需求装载代码协议存在安全缺陷等.

在本文中,我们使用 SPI[18]演算对需求装载代码协议进行分析,并首次发现该协议存在被重放攻击的安全缺陷,通过对缺陷的分析与研究,我们给出了修改后的需求装载代码协议.本文第 1 节主要讨论相关工作,即需求装载代码协议.第 2 节使用 SPI 演算对需求装载代码协议进行分析,并指出其存在的安全缺陷.第 3 节给出一个修改后的强健的需求装载代码协议.最后总结全文.

1 背景知识

ANTS[11]使用一种集成方法建立主动网络结构,为了避免重新为相关报文组装载代码,ANTS 节点 Cache 缓存最近使用的代码.在 ANTS 中,报文被称为一个单元,该单元传输和代码碎片相关的参数.如果一个含有相关代码的报文通过节点时,节点用报文的参数值初始化代码,然后再执行这些代码.反之,如果这些代码不在这个节点上,这个节点就向最近的上游节点请求代码. ANTS 协议是通过 3 个关键向量来完成这些目标的:用 ANTS 的单元代替传统的报文;路由器和终端节点用主动节点代替,在主动节点上,执行和处理通常事务和维持它们相关状态;用代码分配机制来确保处理事务能够自动和动态地分配到它所需要的这些节点上.ANTS[11]和 PAN[14]代码分配机制,采用了需求装载代码协议,如图 1 所示.

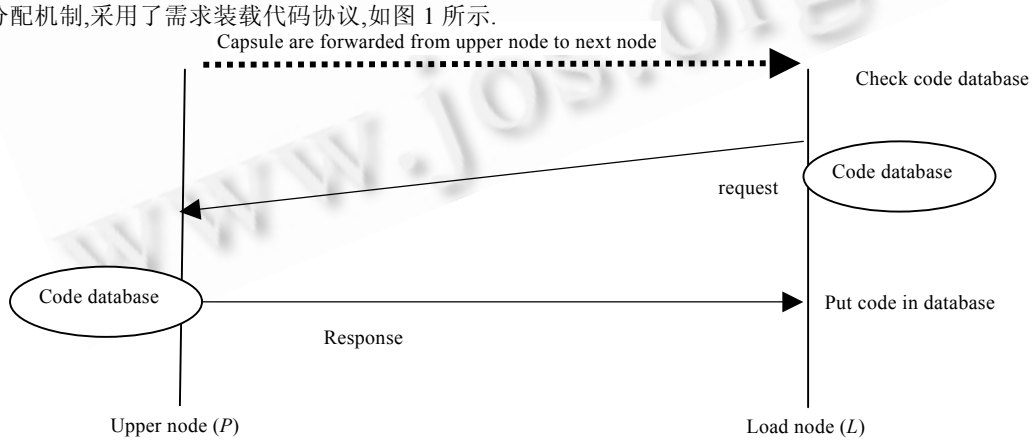

Fig.1 process of demand code laoding

图 1 需求装载代码过程

执行过程如下:

单元识别它们的类型和协议,知道它该走哪条路径.

当一个单元到达一个节点时,检查协议的代码缓存,如果所需要的代码不完全存在,一个基于单元类型和协议装载丢失部分请求向上游节点发出.

当一个节点接收到一个装载请求时,它立即发出装载响应,这个响应含有请求所要求的代码.

当节点接收到装载响应时,它将代码放入缓存,如果这时候所有的代码都在,它唤醒睡眠的单元.如果请求的响应不能被接收到,则睡眠的单元将被丢弃.

2 需求装载代码协议分析

在分析该协议之前,先简单介绍 SPI 演算的基本句法:$\{M\}_N$::=共享密钥加密,$\{M\}_N$ 表示明文用共享密钥,N 进行加密成密文.case L of $\{x\}_N$ in P::=共享密钥解密,表示 L 是密文,试用 N 来解密 L.我们分析图 1 中的需求装载代码协议过程.考虑两个实体,即需求装载主动节点和上游主动节点,分别定义为节点 L 和节点 P.当单元从上游节点到达一个主动节点,检查发现该节点没有所需要的代码时,该节点(我们称为需求装载节点 L)和上游节点 P 之间的需求装载代码交换过程如下:

请求装载:节点 $L \to P:\langle type\_id, protocol\_id, code\_req, \{others\}_{k_{pl}}\rangle$ on C_{pl}.

装载响应:节点 $P \to L:\langle type\_id, protocol\_id, \{code\}_{k_{pl}}\rangle$ on C_{pl}.

我们用 SPI 演算描述节点 P 和 L,如下所示:

$$L(type\_id, protocol\_id) \stackrel{\Delta}{=} (\nu K_{pl})\bar{C}_{pl}\langle type\_id, protocol\_id, code\_req, \{others\}_{k_{pl}}\rangle$$
$$.C_{pl}(type\_id, protocol\_id, x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{pl}}\ \text{in}\ F(type\_id, protocol\_id, y),$$

$$P \stackrel{\Delta}{=} (\nu K_{pl})C_{pl}(type\_id, protcol\_id, code\_req, x'_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{pl}}\ \text{in}\ \bar{C}_{pl}$$
$$\langle type\_id, protocol\_id, \{code\}_{k_{pl}}\rangle.$$

$F(type\_id, protocol\_id, y)$ 表示需求装载节点接收到上游节点所响应的代码后,将执行一系列的动作.需求装载代码协议的整个过程可以描述如下:

$$inst(type\_id, protocol\_id) \stackrel{\Delta}{=} (\nu k_{pl})(L(type\_id, protocol\_id) \mid P).$$

根据 SPI 演算的理论,我们可以用下面的规范说明来论证这个协议的描述是正确的.

$$L(type\_id, protocol\_id)_{specificaton} \stackrel{\Delta}{=} (\nu K_{pl})\bar{C}_{pl}\langle type\_id, protocol\_id, code\_req, \{others\}_{k_{pl}}\rangle$$
$$.C_{pl}(type\_id.protocol\_id, x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{pl}}\ \text{in}\ F(type\_id, protocol\_id, Code).$$

$P_{specification}$ 与 P 一样,即

$$P_{specificaton} \stackrel{\Delta}{=} (\nu K_{pl})C_{pl}(type\_id, protcol\_id, code\_req, x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{pl}}\ in\ \bar{C}_{pl}$$
$$\langle type\_id, protocol\_id, \{code\}_{k_{pl}}\rangle.$$

$$inst_{specification}(type\_id, protocol\_id) \stackrel{\Delta}{=} (\nu k_{pl})(L(type\_id, protocol\_id)_{specification} \mid P_{specification}).$$

上面我们只考虑了两个主动节点之间的一次需求装载代码过程.在实际中会存在多个节点多次需求装载过程.我们定义需求装载协议的主要元素为$\langle i,j,type_id,protocal_id\rangle$,$i,j$ 分别表示源地址和目的地址,$type\_id$,$protocal\_id$ 还是分别表示类型和协议.为了便于描述,将 $\{type\_id, protocol\_id\} \stackrel{def}{=} M$.

我们考虑多个节点多次需求装载的情况,重写上述需求装载代码过程,其形式如下:

$$L(i,j,M) \stackrel{\Delta}{=} (\nu K_{ij})\bar{C}_{ij}\langle M, code\_req, \{others\}_{k_{ij}}\rangle.C_{ij}(i,j,M,x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M,y),$$

$$P\{j\} \stackrel{\Delta}{=} (\nu K_{ij})C_{ij}(M, code\_req, x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ \bar{C}_{ij}\langle M, \{code\}_{k_{ij}}\rangle.$$

为了方便描述,我们设 $(i,j,M) \stackrel{def}{=} \Psi$,则需求装载代码协议过程描述为

$$inst(\Psi_1 ... \Psi_n) \stackrel{\Delta}{=} (\nu k_{ij})((L(\Psi_1) \mid ... \mid L(\Psi_n)) \mid (P_1 \mid ... \mid P_n)).$$

同样,规范上述形式如下:

$$L(i,j,M)_{specfication} \overset{\Delta}{=} (\nu K_{ij})\bar{C}_{ij}\langle M, code\_req, \{others\}_{k_{ij}}\rangle.C_{ij}(i,j,M,x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M,code),$$

$$P(j)_{specification} \overset{\Delta}{=} (\nu K_{ij})C_{ij}(M, code\_req, x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ \bar{C}_{ij}\langle M, \{code\}_{k_{ij}}\rangle,$$

$$inst(\Psi_1 \cdots \Psi_n)_{specificaton} \overset{\Delta}{=} (\nu k_{ij})((L(\Psi_1)_{specificaton} | ... | L(\Psi_n)_{specificaton}) | (P_{specificaton\ 1} | ... | P_{specificaton\ n})).$$

分析需求装载代码的协议见表 1,存在着 4 种可能的重放攻击(因为在主动网络系统,请求装载的是主动代码,这些主动代码能够在主动节点上运行,影响主动节点和执行环境的状态.不同于传统的网络报文,是被网络设备存储-转发,主动报文是计算-存储-转发).

第 1 种重放攻击:攻击者截获 L 节点发出的请求,间隔一段时间重放给上游节点 P,由于 P 不能判断这个重放的请求是否无效,只能执行响应操作.

$$\bar{C}_{ij}\langle M, \{code\}_{k_{ij}}\rangle,$$

而节点 L 收到上游节点 P 的响应之后,也不能判断这个响应是否有效,

$$C_{ij}(M, x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M,y).$$

根据协议,将其代码存入缓存,如果缓存中存在同样的代码则丢弃该代码,但是如果攻击者进行大规模这样的重放攻击,就会使节点 L 的处理资源耗尽或性能降低.如果攻击者采用第 2 种重放攻击,这时候节点 L 和受第一种重放攻击一样,其效率降低,资源耗尽.

对于第 3 种重放攻击,由于攻击者可能修改了 type_id 或 protocol_id,诱骗了上游节点 P 发出响应 $\bar{C}_{ij}\langle M_{new}, \{code_{new}\}_{k_{ij}}\rangle$,这个响应里所含的代码是和被修改的类型或协议相一致的,节点 L 接收到上游节点 P 发出的响应后执行

$$C_{ij}(M, x_{cipher}).case\ x_{cipher}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M_{new}, y_{new}).$$

节点 L 将其放入缓存,等代码全部到齐,再判断是否有相应的单元存在.如果没有相应的单元存在,则丢弃.但由于缓存是有限的,如果受大规模攻击,节点 L 会降低效率和耗尽其处理资源.

攻击者采用第 4 种方式攻击,因为攻击者截获上游节点发出的响应修改其类型或协议,代码保持不变,重放该响应给节点 L,它将执行

$$C_{ij}(M, x_{ciphe}).case\ x_{ciphe}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M_{new}, y).$$

在这种情况下,除了具有上述 3 种重放攻击相同的危害外,还具有最严重的一种危害:当节点 L 在收到修改类型或协议重放的上游节点响应的时候,切好这时候节点 L 刚刚发出和修改类型或协议一样的代码装载请求,正在处于等待节点 P 的响应.那么节点 L 就会误认为重放的响应含的代码是自己所需求的代码,就会放入缓存唤醒单元去执行该代码,执行结果会对主动节点或环境 EE 产生难以预测的后果.

Table 1 Four kinds of replay attack

表 1 4 种重放攻击

Attacker	P (upper node)	L (load node)
(1) Replay load request from node L	$\bar{C}_{ij}\langle M, \{code\}_{k_{ij}}\rangle$	$C_{ij}(M, x_{ciphe}).case\ x_{ciphe}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M,y)$
(2) Replay response from upper node P		$C_{ij}(M, x_{ciphe}).case\ x_{ciphe}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M,y)$
(3) Revised type_id, protocol_id and replay request from node L	$\bar{C}_{ij}\langle M_{new}, \{code_{new}\}_{k_{ij}}\rangle$	$C_{ij}(M, x_{ciphe}).case\ x_{ciphe}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M_{new}, y_{new})$
(4) Revised type_id, protocol_id and replay response from node P		$C_{ij}(M, x_{ciphe}).case\ x_{ciphe}\ \text{of}\ \{y\}_{k_{ij}}\ \text{in}\ F(M_{new}, y)$

通过上述分析我们得知,该协议存在严重的重放攻击危险.由于协议设计人员疏忽,没有考虑到在主动网络环境情况下,受到的重放攻击的后果与传统网络环境下受重放攻击后果是截然不同的.在传统网络环境下,受到重放攻击时,网络设备只是增加了转发或丢弃.而主动网络环境下,主动代码像一段程序那样会在主动节点上运行,产生的影响和后果难以预测.

3 安全需求装载代码协议

本节主要通过增加 nonce(增加的随机数)和握手来保护安全需求装载代码协议,使其能够防止被重放攻击.修改后的协议如图 2 所示.

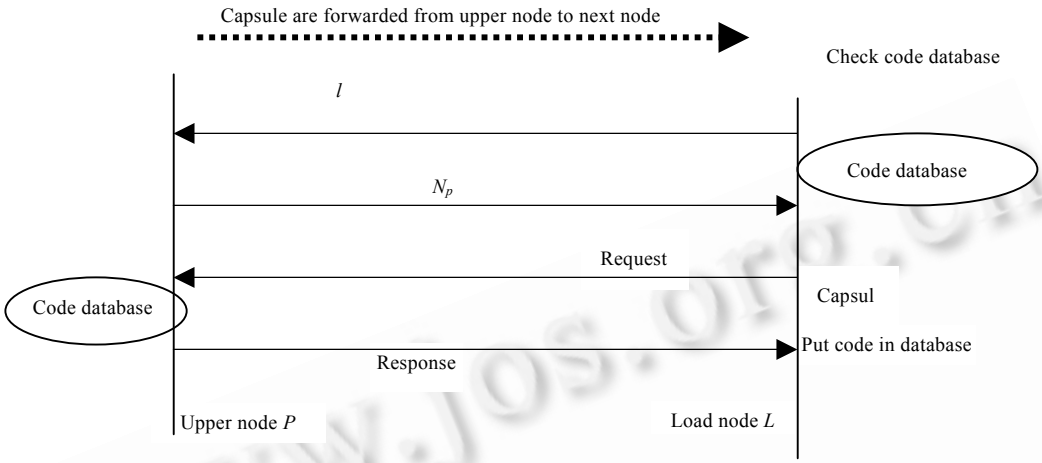

Fig.2 The revised process of demand code loading

图 2 修改后的需求装载代码过程

修改后的协议分成 3 部分,即挑战过程、装载代码请求过程和上游主动节点响应过程.挑战过程由两个过程组成,第 1 个过程是由装载节点 L 向上游节点 P 发出挑战请求 l,第 2 个过程是上游节点 P 产生一个挑战 N_p,这个 N_p(nonce),l 都是随机的、,先前没有被使用过的整数.

我们详细描述这个协议过程:

挑战请求: $L \to P:\langle l\rangle \text{ on } C_{pl}$;

挑战相应: $P \to L:\langle N_p\rangle \text{ on } C_{pl}$;

代码装载请求: $L \to P:\langle type\_id, protocol\_id, code\_req, l, \{l, N_p\}_{k_{pl}}\rangle \text{ on } C_{pl}$;

代码装载响应: $P \to L:\langle type\_id, protocol\_id, N_p, \{l, N_p, code\}_{k_{pl}}\rangle \text{ on } C_{pl}$.

重写修改后的协议 L 节点、P 节点和整个协议过程的 SPI 演算,形式如下:

$$L(i,j,M)\overset{\Delta}{=}(vK_{ij})\bar{C}_{ij}(l).C_{ij}(x_{nonce}).\bar{C}_{ij}\langle M, code\_req, l, \{l, x_{nonce}\}_{k_{ij}}\rangle$$
$$.C_{ij}(M, x'_{nonce}, x'_{cipher})[x'_{nonce} \text{ is } x_{nonce}].case\ x'_{cipher} \text{ of } \{x'_{l}, y''_{nonce}, y\}_{k_{ij}}.[x'_l \text{ is } l][y''_{nonce} \text{is } x_{nonce}].\text{in } F(M,y),$$

$$P(j)\overset{\Delta}{=}(vK_{ij})C_{ij}(x_l).\bar{C}_{ij}(N_p).C_{ij}(M, code\_req, x'_l, x_{cipher}).[x'_l \text{ is } x_l]case\ x_{cipher} \text{ of}$$
$$\{y_l, y_{nonce}\}_{k_{ij}}.[y_l \text{ is } x_l][y_{nonce} \text{ is } N_p]\text{in } \bar{C}_{ij}\langle M, N_p, \{y_l, N_{p,} code\}_{k_{ij}}\rangle.$$

我们定义 $(i,j,M)\overset{def}{=}\Psi$,

$$inst_{new}(\Psi_1 ... \Psi_n)\overset{\Delta}{=}(vk_{ij})((L(\Psi_1)|...|L(\Psi_n))|(P(j)_1|...|P(j)_n)),$$

修改后的需求装载协议过程的规范化说明和上节定义基本相似,在此省略.

我们同样使用 4 种重放攻击方法来分析修改后的协议安全性,其分析结果见表 2.

通过表 2 可以看出,在第 1 种重放攻击情况下,因为上游节点 P 判断攻击者重放请求中 l 和 N_p 与自己存有的 l 和 N_p 是否相同,如果不相同,则丢弃,这样就避免了对重放请求的响应(因为上游节点 P 完成对节点 L 请求的响应后,将丢弃原先和节点 L 协议的 l 和 N_p,所以重放攻击时,节点中 N_p 和 l 已经和以前不一样了.同时,由于 l 和 N_p 是随机的,这样就保证了协议的安全性).对于第 2 种重放攻击,由于节点 L 对重放响应中的 N_p 和 l 进行判

断,再进行一系列的处理,因为节点 L 中的 l 和 N_p 已经和重放响应中的 l 和 N_p 不一样,所以这个重放响应被直接丢弃.第 3、4 种重放攻击情况和第 1、2 种相类似,因为节点 L 和 P 通过判断接收到 l 和 N_p 与现有的 N_p 和 l 不一致,丢弃修改类型或协议的响应和请求,这样就阻止了第 3、4 种重放攻击.

Table 2 Results of protocol security analysis

表 2 协议安全性分析结果

Attacker	P(upper node)	L(load node)
(1) Replay load request from node L	$C_{ij}(M, code\_req, x'_l, x_{cipher}).[x'_l \text{ is } x_l]case$ x_{cipher} of $\{y_l, y_{nonce}\}_{k_{ij}}.[y_l \text{ is } x_l][y_{nonce} \text{ is } N_p]$	The node L can prevent attack from replay
(2) Replay response from upper node P		$C_{ij}(M, x'_{nonce}, x'_{cipher})[x'_{nonce} \text{ is } x_{nonce}].case$ x'_{cipher} of $\{x'_l, y''_{nonce}, y\}_{k_{ij}}.[x'_l \text{ is } l]$ $[y''_{nonce} \text{ is } x_{nonce}].in\ F(M, y)$
(3) Revised type_id, protocol_id and replay request from node L	$C_{ij}(M, code\_req, x'_l, x_{cipher}).[x'_l \text{ is } x_l]case$ x_{cipher} of $\{y_l, y_{nonce}\}_{k_{ij}}.[y_l \text{ is } x_l][y_{nonce} \text{ is } N_p]$	The node L can prevent attack from replay
(4)revised type_id, protocol_id and replay response from node P		$C_{ij}(M, x'_{nonce}, x'_{cipher})[x'_{nonce} \text{ is } x_{nonce}].case$ x'_{cipher} of $\{x'_l, y''_{nonce}, y\}_{k_{ij}}$ $.[x'_l \text{ is } l][y''nonce''_{nonce} \text{ is } x_{nonce}]in\ F(M, y)$

4 结束语

由于主动网络是存储-计算-转发模型,不同于传统网络的存储-转发模型,在传统网络中,当网络设备遇到重放攻击时,其影响很小,一般只是增加了网络设备的转发量,但对主动网络来说,当其受到重放攻击的时候,由于主动报文含有主动代码,该代码和程序一样能在主动节点上进行执行,影响主动节点及其 EE 的状态,对主动节点产生难以预测的后果,同时也降低了主动节点的处理效率和性能.

本文使用 SPI 演算对需求装载代码协议进行分析,发现其存在被重放攻击的安全漏洞.攻击者能够通过重放攻击对主动网络节点产生难以预测的危害,为了消除该协议被重放攻击的危险,我们修改原有需求装载代码协议,并增加了其阻止重放攻击的能力.

References:

[1] Xia ZY, Zhang SY. Survey of active network research. Mini-Microsystems, 2003,24(10):1821−1824 (in Chinese with English abstract).

[2] Tennenhouse DL, Smith JM, Sincoskie WD, Wetherall DJ, Minden G. A survey of active network research. IEEE Communication Magazine, 1997,35(1):80−86.

[3] Tennhouse DL, Wetherall DJ. Towards an active network architecture. Computer Communication Review, 1996,26(2):464−472.

[4] Calvert KL. Architectural framework for active networks. Version 1.0 University of Kentucky, 1999. http://www.ccgatech.edu/project/canes/papers/arch-1-0.ps.gz

[5] Wetherall DJ, Guttag JV, Tennenhouse DL. ANTS: A toolkit for building and dynamically deploying network protocols. In: IEEE Openarch'98. IEEE, 1998. 117.

[6] Schwartz B, Jackson AW, Strayer WT, Zhou WY, Rockwell RD, Partridge C. Smart packet for active networks. In: Proc. of the 1999 IEEE 2nd Conf. on Open Architectures and Network Programming (OPENARCH'99). 1999.

[7] Nygren EL, Garland SJ, Kaashoek MF. PAN: A high-performance active network node supporting multiple mobile code system. In: Proc. of the 1999 IEEE 2nd Conf. on Open Architecture and Network Programming. 1999.

[8] AN Node OS Working Group. NodeOS interface specification. 2000. http://www.cs.princeton.edu/nsg/papers/nodeos.ps

[9] Xia ZY, Zhang SY, Zhong YP. A kind of security negotiation protocol for active network. In: Proc. of the Conf. the ACM Information Security 2002. 2002. 156−162.

[10] Zhou YZ, Zhang YX. A program transfer protocol for active network. In: Proc. of the Int'l Conf. on Telecommunications 2002. Vol.3, 2002.

[11] Xia ZY, Zhang SY. Design of secure system architecture model for active network. Journal of Software, 2002,13(8):1352−1360. http://www.jos.org.cn/1000-9825/1352.htm

[12] Lindell B. Active networks protocol specification for Hop-By-Hop message authentication and integrity. Draft-nodeos-security-00.txt. 2000. http://www.isi.edu/abone/Documents/Ossec.txt

[13] Liu ZY, Campbell RH. Securing the Node of Active Networks. Kluwer Academic Publishers, 2000.

[14] Campbell RH, Liu ZY. Dynamic interoperable security architecture for active network. In: Proc. of the IEEE Openarch 2000. 2000. 32−41.

[15] Lu YM, Qian DP, Xu Bin, Wang L. Execution environments for active network based on programmable mobile soft devices. Journal of Software, 2002,13(2):227−234 (in Chinese with English abstract). http://www.jos.org.cn/ 1000-9825/13/1352.pdf

[16] Alexander DS, Braden B, Gunter CA, Jackson AW, Keromytis AD, Minden GJ, Wetherall D. Active network encapsulation protocol (ANEP). 1997. http://www.cis.upenn.edu/switchware/ANEP/docs/ANEP.txt

[17] Wetherall D, Tennenhouse D. Active IP Option. In: Proc. of the 7th ACM SIGOPS European Workshop ACM. 1996.

[18] Abadi M, Gordon A. A calculus for cryptographic protocols: The SPI calculus. Information and Computation, 1999,148(1):1−70.

附中文参考文献:

[11] 夏正友,钟亦平,张世永.Active Network 研究综述.小型微型计算机系统,2003,24(10):1821−1824.

[15] 陆月明,钱德沛,徐斌,王磊.基于可编程移动软设备的主动网络执行环境.软件学报,2002,13(2):227−234. http://www.jos.org.cn/1000-9825/13/1352.pdf