# strongswan SA分析（一）

## 1 概念

下面主要介绍两个本文将要阐述的核心概念。他们是SA和SP。注意，这不是一篇不需要背景知识的文章。作者认为你适合阅读接下来内容的的前提是，你已经具备了一下三方面的知识：

* a. 什么是VPN。
* b. 什么是IPsec，包括IKE，ESP，strongswan都是什么等。
* c. 一般的linux使用方法和常见概念。

### 1.1 什么是SAD，SPD

SAD是Security Association Database的缩写。  
SPD是Security Policy Database的缩写。  
SAD是用来存储SA的数据库。SPD是用来存储SP的数据库。

### 1.2 什么是SPI

SPI是Security Parameter Index的缩写。是有一组数字（长度？）。被使用在SAD和SPD里作为索引的一部分。是由IKE协商的两侧客户端随机选择的UUID？。0-255是被保留的值，禁止在SPI中使用。

### 1.3 什么是SA

SA是Security Association的缩写。SA是一组算法和算法参数（包括key）的集合，用来完成单个方向的数据流加密和验证任务。通过SPI加数据包的目的地址可以唯一查找到一个SA。

包含的属性：

* 加密算法
  + 属性
  + key
* 验证算法
  + 属性
  + key
* SPI
* 目的地址

### 1.4 什么是SP

SP是Security Policy的缩写。SP是一条规则，决定一条流（flow）是否需要被IPsec处理。SP的处理有三种方式：

* 丢弃
* 不处理
* 处理

需要被IPsec处理的流，会被指向到一个template。一个template可以理解为指向一个SA，template包含以下属性：

* 协议
  + AH或ESP。
* 模式
  + transport或tunnel模式。
* pattern
  + 源IP加目的IP对。
  + NAT的PORT对。

SP有一个方向属性，取值分别为：

* out
* in
* fwd

### 1.5 总结

在整个IPsec的数据流转逻辑中，SP用来表达What todo。SA用来表达How todo。

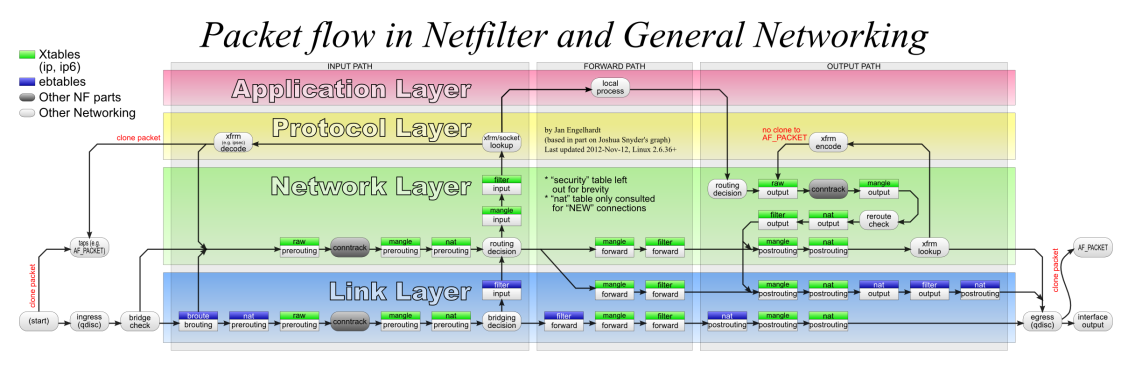
## 2 数据流

简单的说。明文报在通过IPsec VPN设备变成ESP发出去的过程是：

1. 查找路由。
2. 查找policy决定是否需要被ESP
3. 查找SA并加密封装。
4. 加密封装后的包再查路由。

IPsec报在通过IPsec VPN设备变成非加密包发出去的过程：

1. 查找路由。
2. 查找policy决定是否需要要解ESP
3. 查找SA并解密解封装。
4. 解密解封装后的包再查路由。



### 2.1 举个栗子

路由

[root@T9 sbin]# ip route

default via 192.168.7.1 dev eth0 proto static metric 100 10.129.0.0/24 dev eth1 proto kernel scope link src 10.129.0.1 metric 100 192.168.7.0/24 dev eth0 proto kernel scope link src 192.168.7.129 metric 100

Policy

[root@T9 sbin]# ip xfrm policy

src 10.9.0.0/16 dst 10.129.0.0/16

dir fwd priority 383616 ptype main

tmpl src 192.168.7.9 dst 192.168.7.129

proto esp reqid 1 mode tunnel

src 10.9.0.0/16 dst 10.129.0.0/16

dir in priority 383616 ptype main

tmpl src 192.168.7.9 dst 192.168.7.129

proto esp reqid 1 mode tunnel

src 10.129.0.0/16 dst 10.9.0.0/16

dir out priority 383616 ptype main

tmpl src 192.168.7.129 dst 192.168.7.9

proto esp reqid 1 mode tunnel

sa

[root@T9 sbin]# ip xfrm statesrc 192.168.7.129 dst 192.168.7.9

proto esp spi 0xc42ac7f3 reqid 1 mode tunnel

replay-window 0 flag af-unspec

auth-trunc hmac(sha256) 0x5f7b99e.....eb20948fb2f8fc713caf2d43b4 128

enc cbc(aes) 0x48144872d5f4f9a6a762b68785e6f265src 192.168.7.9 dst 192.168.7.129

proto esp spi 0xc1c8ad99 reqid 1 mode tunnel

replay-window 32 flag af-unspec

auth-trunc hmac(sha256) 0x7efc5d2172.....0c0dedf053b0b6ae5aa2f012 128

enc cbc(aes) 0x808efcfaa45a543b69efe08158accaa3

## 3 理解linux kernel中的sa概念和管理

### 3.1 提供给用户的sa接口

理解kernel sa对用户展示的形态，可以帮助我们理解linux kernel对于ipsec sa的建模和抽象。对我们在VPN产品的sa模块设计中将提供帮助。

#### 3.1.1 使用racoon配置sa

setkey add 192.168.0.1 192.168.1.2 esp 0x10001

-m tunnel

-E des-cbc 0x3ffe05014819ffff

-A hmac-md5 "authentication!!"

从以上信息可以很容易开始各个参数表达的含义，其中-E代表加密算法和它的key，-A代表验证算法和它的key。0x10001为spi。

#### 3.1.2 使用racoon配置policy

setkey spdadd 10.0.11.41/32[21] 10.0.11.33/32[any] any

-P out ipsec esp/tunnel/192.168.0.1-192.168.1.2/require

第一行代表五元组，any代表协议。第二行代表policy的具体描述：方向，action，template。

#### 3.1.3 总结

通过以上两个小节的描述，读者应该已经很容易的总结出了配置一个SA和一个policy所需要提供的最基本的信息了。作者将在本章的最后，对sa和poliyc所包含的所有必须信息进行一个统一的总结。  
另外，通过上文的语法，我们应该能够发现，policy与sa之间的match操作，是需要一个稍负责的匹配逻辑来实现的，而不仅仅是一个简单的匹配关系。

### 3.2 netlink的SA接口

strongswan是目前使用两种方式与内核进行ipsec的配置交互，分别为netlink和pfkey。如官方文档所述，netlink是strongswan默认启用的，变成stable的接口方式。整个调研工作也是以netlink方式为出发点展开的，现简单介绍如下。

#### 3.2.1 什么是netlink

netlink是复用了socket方式的内核与用户态IPC方法。  
这里有一篇写的非常好的文章，讲[netlink为什么会产生](https://www.linuxjournal.com/article/7356)。由于人家写的实在是太好了，我已经没有什么可写的了，只能做个概要，如下：

因为作者图画是业余的，所以看懂的这个概要图的前提是，你必须懂得BSD socket的api如何使用。

#### 3.2.2 接口方式

用netlink方式配置ipsec的方法。

##### netlink的一般用法

###### 初始化socket

与常规的socket用法相同，只是传入参数是netlink定义的特有参数。

int socket(int domain, int type, int protocol)bind(fd, (struct sockaddr\*)&nladdr, sizeof(nladdr));

###### 下发配置信息到kernel

使用socket的标准send，write接口将特定格式的参数下发给kernel。  
参数格式如下：

struct nlmsghdr

{

\_\_u32 nlmsg\_len; /\* Length of message \*/

\_\_u16 nlmsg\_type; /\* Message type\*/

\_\_u16 nlmsg\_flags; /\* Additional flags \*/

\_\_u32 nlmsg\_seq; /\* Sequence number \*/

\_\_u32 nlmsg\_pid; /\* Sending process PID \*/

};

这个参数结构体是传入参数的头部，紧接着这个头部之后的内存是真正的参数的值。它的解析方法由nlmsg\_type的值来确定。它的结尾由nlmsg\_len的数值来决定。

##### 添加sa

添加sa的时候，nlmsghdr后面的参数为结构体

struct xfrm\_usersa\_info

nlmsg\_type的值为：XFRM\_MSG\_NEWSA  
这部分内容定义在系统文件下：

/usr/include/linux/xfrm.h

这个结构体后边，还需要追加算法部分的信息，如下：

struct xfrm\_algostruct xfrm\_algo\_auth

##### 添加policy

添加policy的时候，nlmsghdr后面的参数为结构体

struct xfrm\_userpolicy\_info

nlmsg\_type的值为：XFRM\_MSG\_NEWPOLICY  
这部分内容定义在系统文件下：

/usr/include/linux/xfrm.h

### 3.3 xfrm的SA接口

#### 3.3.1 什么是xfrm

[xfrm](http://man7.org/linux/man-pages/man8/ip-xfrm.8.html)(transform)是一个IP包转发框架。主要实现以下三部分功能：

* IPsec protocol suite
* IP Payload Compression Protocol
* Mobile IPv6

#### 3.3.2 内核代码

linux/net/xfrm/

主要函数

Xfrm\_lookup() xfrm lookup(SPD and SAD) methodXfrm\_input() xfrm processing for an ingress packetXfrm\_output() xfrm processing for an egress packetXfrm4\_rcv() IPv4 specific Rx methodXfrm6\_rcv() IPv6 specific Rx methodEsp\_input() ESP processing for an ingress packetEsp\_output() ESP processing for an egress packetAh\_output() AH processing for an ingress packetAh\_input() ESP processing for an egress packetxfrm\_policy\_alloc() allocates an SPD objectXfrm\_policy\_destroy() frees an SPD objectxfrm\_ policy\_lookup SPD lookupxfrm\_policy\_byid() SPD lookup based on idXfrm\_policy\_insert() Add an entry to SPDXfrm\_Policy\_delete() remove an entry from SPDXfrm\_bundle\_create() creates a xfrm bundleXfrm\_policy\_delete() releases the resources of a policy objectXfrm\_state\_add() add an entry to SADXfrm\_state\_delete() free and SAD objectXfrm\_state\_alloc() allocate an SAD objectxfrm\_state\_lookup\_byaddr() src address based SAD lookupxfrm\_state\_find() SAD look up based on dstxfrm\_state\_lookup() SAD lookup based on spi

#### 3.3.3 API

api文件

include/uapi/linux/xfrm.h

主要的API

XFRM\_MSG\_NEWSA To add a new SA to SAD

XFRM\_MSG\_DELSA To delete a new SA to SAD

XFRM\_MSG\_GETSA To get a new SA to SAD

XFRM\_MSG\_FLUSHSA To flush SAD

XFRM\_MSG\_NEWPOLICY To add a new policy to SPD

XFRM\_MSG\_DELPOLICY To delete a new policy to SPD

XFRM\_MSG\_GETPOLICY To get a new policy to SPD

XFRM\_MSG\_FLUSHPOLICY To flush SPD

#### 3.3.4 sa的传入参数

struct xfrm\_usersa\_info {

struct xfrm\_selector sel; // 被加密网段？为啥要有这个？

struct xfrm\_id id; // 目的ip，spi，协议ah/esp

xfrm\_address\_t saddr; // 源ip

struct xfrm\_lifetime\_cfg lft;

struct xfrm\_lifetime\_cur curlft;

struct xfrm\_stats stats;

\_\_u32 seq;

\_\_u32 reqid;

\_\_u16 family;

\_\_u8 mode; // transport / tunnel

\_\_u8 replay\_window;

\_\_u8 flags;

};

算法参数是追加在SA结构体之后的内存块，根据不同的类型决定不同的结构。示例：

struct xfrm\_algo {

char alg\_name[64];

unsigned int alg\_key\_len; /\* in bits \*/

char alg\_key[0];

};

struct xfrm\_algo\_auth {

char alg\_name[64];

unsigned int alg\_key\_len; /\* in bits \*/

unsigned int alg\_trunc\_len; /\* in bits \*/

char alg\_key[0];

};

#### 3.3.5 policy的传入参数

struct xfrm\_userpolicy\_info {

struct xfrm\_selector sel; //网段：ip，port，协议

struct xfrm\_lifetime\_cfg lft;

struct xfrm\_lifetime\_cur curlft;

\_\_u32 priority; //

\_\_u32 index;

\_\_u8 dir; //方向：in out fwd

\_\_u8 action; // allow, block

\_\_u8 flags;

\_\_u8 share;

};

## 4 xfrm的实现

### 4.1 用于存储sa的内部数据结构

struct xfrm\_state {#ifdef CONFIG\_NET\_NS

struct net \*xs\_net;#endif

union {

struct hlist\_node gclist;

struct hlist\_node bydst;

};

struct hlist\_node bysrc;

struct hlist\_node byspi;

atomic\_t refcnt;

spinlock\_t lock;

struct xfrm\_id id;

struct xfrm\_selector sel;

struct xfrm\_mark mark;

u32 tfcpad;

u32 genid;

/\* Key manager bits \*/

struct xfrm\_state\_walk km;

/\* Parameters of this state. \*/

struct {

u32 reqid;

u8 mode;

u8 replay\_window;

u8 aalgo, ealgo, calgo;

u8 flags;

u16 family;

xfrm\_address\_t saddr;

int header\_len;

int trailer\_len;

u32 extra\_flags;

} props;

struct xfrm\_lifetime\_cfg lft;

/\* Data for transformer \*/

struct xfrm\_algo\_auth \*aalg;

struct xfrm\_algo \*ealg;

struct xfrm\_algo \*calg;

struct xfrm\_algo\_aead \*aead;

/\* Data for encapsulator \*/

struct xfrm\_encap\_tmpl \*encap;

/\* Data for care-of address \*/

xfrm\_address\_t \*coaddr;

/\* IPComp needs an IPIP tunnel for handling uncompressed packets \*/

struct xfrm\_state \*tunnel;

/\* If a tunnel, number of users + 1 \*/

atomic\_t tunnel\_users;

/\* State for replay detection \*/

struct xfrm\_replay\_state replay;

struct xfrm\_replay\_state\_esn \*replay\_esn;

/\* Replay detection state at the time we sent the last notification \*/

struct xfrm\_replay\_state preplay;

struct xfrm\_replay\_state\_esn \*preplay\_esn;

/\* The functions for replay detection. \*/

struct xfrm\_replay \*repl;

/\* internal flag that only holds state for delayed aevent at the

\* moment

\*/

u32 xflags;

/\* Replay detection notification settings \*/

u32 replay\_maxage;

u32 replay\_maxdiff;

/\* Replay detection notification timer \*/

struct timer\_list rtimer;

/\* Statistics \*/

struct xfrm\_stats stats;

struct xfrm\_lifetime\_cur curlft;

struct tasklet\_hrtimer mtimer;

/\* used to fix curlft->add\_time when changing date \*/

long saved\_tmo;

/\* Last used time \*/

unsigned long lastused;

/\* Reference to data common to all the instances of this

\* transformer. \*/

const struct xfrm\_type \*type;

struct xfrm\_mode \*inner\_mode;

struct xfrm\_mode \*inner\_mode\_iaf;

struct xfrm\_mode \*outer\_mode;

/\* Security context \*/

struct xfrm\_sec\_ctx \*security;

/\* Private data of this transformer, format is opaque,

\* interpreted by xfrm\_type methods. \*/

void \*data;

};

会被插入两个hash表

1. Hash table by (spi,daddr,ah/esp) to find SA by SPI. (input,ctl)

2. Hash table by (daddr,family,reqid) to find what SAs exist for given

destination/tunnel endpoint. (output)

### 4.2 用于存储sa的内部数据结构

struct xfrm\_policy {#ifdef CONFIG\_NET\_NS

struct net \*xp\_net;#endif

struct hlist\_node bydst;

struct hlist\_node byidx;

/\* This lock only affects elements except for entry. \*/

rwlock\_t lock;

atomic\_t refcnt;

struct timer\_list timer;

struct flow\_cache\_object flo;

atomic\_t genid;

u32 priority;

u32 index;

struct xfrm\_mark mark;

struct xfrm\_selector selector;

struct xfrm\_lifetime\_cfg lft;

struct xfrm\_lifetime\_cur curlft;

struct xfrm\_policy\_walk\_entry walk;

struct xfrm\_policy\_queue polq;

u8 type;

u8 action;

u8 flags;

u8 xfrm\_nr;

u16 family;

struct xfrm\_sec\_ctx \*security;

struct xfrm\_tmpl xfrm\_vec[XFRM\_MAX\_DEPTH];

};

### 4.3 数据结构之间的存储结构

TODO

### 4.4 关键函数

xfrm\_lookup()xfrm\_output()xfrm4\_policy\_check() // 在ipv4中被调用。

## 5 strongswan中的sa

### 5.1 概述

从IKE协议的角度上，有两个SA，一个叫IKE\_SA，一个叫CHILD\_SA。本章讨论的sa，特指下图中的CHILD\_SA。  
本篇文章，通篇讨论的SA指的都是这里的CHILD\_SA。

CHILD\_SA在strongswan的框架里，主要存在与两个部分。

1. IKE协商过程。  
   CHILD\_SA是IKE协商过程中的输出。IKE协商过程结束后，IKE-SA Manager将CHILD\_SA交个strongswan框架。
2. IPsec隧道建立过程。  
   CHILD\_SA是IKE协商过程中的输入。strongswan框架将CHILD\_SA交给libcharon plugin由特定的plugin与kernel通信，在kernel中完成IPsec tunnel的建立过程。
3. IPsec在转发过程。  
   这部分和strongswan的框架没有了关系，由内核完成。

+---------------------------------+ +----------------------------+

| Credentials | | Backends |

+---------------------------------+ +----------------------------+

+------------+ +-----------+ +------+ +----------+

| receiver | | | | | +------+ | CHILD\_SA |

+----+-------+ | Scheduler | | IKE- | | IKE- |--+----------+

| | | | SA |--| SA | | CHILD\_SA |

+-------+--+ +-----------+ | | +------+ +----------+

<->| socket | | | Man- |

+-------+--+ +-----------+ | ager | +------+ +----------+

| | | | | | IKE- |--| CHILD\_SA |

+----+-------+ | Processor |--------| |--| SA | +----------+

| sender | | | | | +------+

+------------+ +-----------+ +------+

+---------------------------------+ +----------------------------+

| Bus | | Kernel Interface |

+---------------------------------+ +----------------------------+

| | |

+-------------+ +-------------+ V

| File-Logger | | Sys-Logger | //////

+-------------+ +-------------+

#### 5.1.1 strongswan中的plugin

上一小节提到了plugin，接下来讲解[plugin](https://wiki.strongswan.org/projects/strongswan/wiki/PluginList)。  
有两类plugins。**一类是libstrongswan的plugin，一类是libcharon的plugin**。  
libstrongswan的plugin主要提供加密，认证，数据库相关的功能。  
libcharon的plugin主要提供“specific needs”。。。我们接下来要讨论的与sa下发相关的plugin都在  
libcharon这一类里。他们包括：

* kernel-libipsec  
  用户态的转发平面，目前还处于高实验性阶段。转发性能没有kernel。主要用来满足不能使用kernel转发的场景。
* kernel-netlink  
  使用netlink接口与linux kernel的xfrm模块交互。目前输出稳定使用阶段，默认首选。
* kernel-iph  
  windows操作系统的接口。
* kernel-pfkey  
  使用pkkey接口与linux kernel的xfrm模块进行交互，高实验性阶段。
* kernel-wfp  
  windows操作系统的接口。

本文，只关心kernel-netlink的plugin。

### 5.2 启动过程

#### 5.2.1 概述

strongswan的启动方式有多种。可以和各种不同的系统对接，包括systemd，networkmanager等。

* starter  
  ipsec命令使用的守护进程。用ipsec start命令，就会启动这个进程。
* charon-nm  
  networkmanager的plugin。什么是nm的plugin？
* charon-systemd  
  按照systemd的daemon style实现的一个进程。由systemd启动。
* charon-svc  
  windows的服务。

各种启动方式的最终目的都是启动最终目的都是启动charon进程。所以，最简的启动方法就是：

* 直接运行charon进程

当然，这种方式没有daemon守护，但是功能完整。

#### 5.2.2 调试方法

如上一小节所述。charon进程可以直接运行。所以调试的时候直接使用gdb运行charon就可以了。

# gdb `which charon`

#### 5.2.3 starter的启动过程

starter的启动方法是通过ipsec脚本执行start命令，这样便启动了strongswan服务。

# ipsec start

##### ipsec脚本

源码位置

strongswan-5.7.1/src/ipsec/\_ipsec

ipsec脚本解析start参数后，会执行如下命令，启动daemon进程starter

${IPSEC\_DIR}/starter --daemon charon

##### starter进程

源码位置

strongswan-5.7.1/src/starter/starter.c

starter的主要功能是启动charon进程，并进行守护。

* daemon的初始工作  
  重定向输出，signal响应等。
* 启动charon
* 加载ipsec.conf中的配置。