MODULE *SPA_Attack*

This is a specification of the SDP architecture and algorithm. The specification is based on the following materials:
https://cloudsecurityalliance.org/artifacts/software-defined- perimeter-zero-trust-specification-v2/ http://www.cipherdyne.org/fwknop/

Author: *Dong.luming@zte.com.cn*

EXTENDS *FiniteSets*, *Sequences*, *Naturals*, *Integers*, *TLC*, *Bitwise*

The end point user's (*SDP* client) configuration, includes local *IP* and account *Info*.

CONSTANT *ClientCfg* @type: [*LoginID* $\mapsto$ String, *Key* $\mapsto$ Integer, *SrcIp* $\mapsto$ Integer];

The *SDP* controller's exposure service info, inludes listening *IP* and port.

CONSTANT *SDPSvrCfg* @type: [*IP* $\mapsto$ Integer, *Port* $\mapsto$ Integer];

The target server's exposure service info, inludes server *IP* and listening port.

CONSTANT *SvrCfg* @type: [*IP* $\mapsto$ Integer, *Port* $\mapsto$ Integer];

The attacker's configuration, inludes local *IP*.

CONSTANT *AttackerCfg* @type: [*SrcIp* $\mapsto$ Integer];

The match any type value for a *ACL Rule*.

CONSTANT *MATCH_ANY* @type: Integer;

For an user's socket link , the random local port range.

CONSTANT *USER_BASEPORT* @type: Integer;

For an attacker's socket link , the random local port range.

CONSTANT *ATTACKER_BASEPORT* @type: Integer;

If the attacker and user are in the same *LAN* with a shared public *IP* for *NAT*.

CONSTANT *NAT_FLAG* @type: *BOOL*;

The invalid authentication session *ID* value.

If a data access link with an invalid authentication session *ID*, it means we don't know the data access is resulted by which *Auth* ses

CONSTANT *UNKNOWN_AUTH_ID* @type: Integer;

If the legal user and attacker are in the same *LAN* with shared public *IP*, then the local port range after *SNAT* must not conflict w

ASSUME $(NAT\_FLAG = \text{TRUE} \Rightarrow AttackerCfg.SrcIp = ClientCfg.SrcIp \land USER\_BASEPORT \neq ATTACKEF$

ASSUME $(SDPSvrCfg.IP \neq ClientCfg.SrcIp \wedge SDPSvrCfg.IP \neq AttackerCfg.SrcIp)$

ASSUME $(SvrCfg.IP \neq ClientCfg.SrcIp \wedge SvrCfg.IP \neq AttackerCfg.SrcIp)$

ASSUME $(SvrCfg.IP \neq SDPSvrCfg.IP)$

The variables related to legal user's state machine

The legal user's status indicates which session it is undergoing now.

VARIABLE $uState$ @type: {"Start_Auth", "Auth_End", "Connecting", "Connected"};

The legal user's IP address get from configuration.

VARIABLE uIP @type: Integer;

The legal user's ID for authentication.

VARIABLE uID @type: String;

The legal user's Secret Key for authentication.

VARIABLE Key @type: Integer;

The legal user's $Sync$ counter value (Time Stamp) for SDP authentication, the counter increases randomly each auth s

VARIABLE $uTstamp$ @type: Integer;

The legal user's knowledge for SDP controller's info got from configuration.

VARIABLE $uSDPSvrInfo$ @type: [$IP \mapsto$ Integer, $Port \mapsto$ Integer];

The legal user's knowledge for target server's info got from configuration.

VARIABLE $uSvrInfo$ @type: [$IP \mapsto$ Integer, $Port \mapsto$ Integer];

The legal user's TCP links connected with target server for access.

VARIABLE $uTCPLinkSet$ @type: Set([$sIP \mapsto Integer$,
$sPort \mapsto$ Integer,
$dIP \mapsto$ Integer,
$dPort \mapsto$ Integer,
$State \mapsto$ {"SYN_SENT", "ESTABLISHED"}]);

The legal user's $Authenticaiton$ sessions in history recorded in $Log.Each$ session identified by a SPA message.

VARIABLE $uAuthSession$ @type: Set([$MsgID \mapsto$ "SPA_AUTH",
$sIP \mapsto uIP$,
$sPort \mapsto RandomPort(uTstamp, USER\_BASEPORT)$,
$dIP \mapsto uSDPSvrInfo.IP$, \ ∗The SDP Controller's IP and port for SPA protocol

$dPort \mapsto uSDPSvrInfo.Port$,
$ClientID \mapsto uID$, $Tstamp \mapsto uTstamp$, \ * increased each session to anti Replay
$SvrIP \mapsto Encrypt(uSvrInfo.IP, Key)$, \ * Target Server's exposure service $Info$, need to kept secret
$SvrPort \mapsto Encrypt(uSvrInfo.Port, Key)$,
$HMAC \mapsto CalcHMAC(uIP, uID, uTstamp, Encrypt(uSvrInfo.IP, Key), Encrypt(uSvrInfo.Port, Key), Ke$
, \ * $HMAC$ of payload
$Type \mapsto Set(\text{"User"}, \text{"Attacker"})]$); \ * Flag to indicate this message is built by legal user or attacker \ * this flag not invloved in inter-operation between SDP protocol entities,only for statistic

The legal user equipment's packets channnel for recieving data plane packets, corresponds to its physical NIC.

VARIABLE $uChannel$ @type: Sequence of TCP Packets $Seq([sIP \mapsto p.dIP, \backslash * TCP\ packets\ for\ data\ access, for\ this\ model$

$sPort \mapsto p.dPort$, \ * simulate the data plane access stream only by TCP connection proceudre
$dIP \mapsto p.sIP$, \ * IE. if user establish a TCP connection with target server, that
$dPort \mapsto p.sPort$, \ *means a sucessful data access session.
$Flg \mapsto Set(\text{"TCP\_SYN"}, \text{"TCP\_SYN\_ACK"}, \text{"TCP\_ACK"})$,
\ * TCP handshake packets type.
$Type \mapsto Set(\text{"User"}, \text{"Attacker"})]$; \ * Flag to indicate this access is initiated by legal user or attacker \ * this flag not invloved in inter-operation between SDP protocol entities,only for statistic

The legal $User$'s private variables ($uChannel$ is public variable of user, for other entity can operate and modify $uChannel$ variable c

$user\_vars \triangleq \langle uState, uIP, uID, Key, uTstamp, uSDPSvrInfo, uSvrInfo, uTCPLinkSet, uAuthSession \rangle$

The variables related to SDP Server's (SDP Controller) state machine

The SDP controller's status indicates this entity's service is available or faulty.

VARIABLE $SDPSvrState$ @type: $Set(\text{"Work"})$

The SDP controller successfully processed $Auth$ sessions in history recorded in Log.

VARIABLE $SDPSucSession$ @type: $uAuthSession$

The Legal user accounts info recorded in SDP controller's IAM system.

VARIABLE $Account$ @type: $Set([ClientID \mapsto ClientCfg.LoginID,$
$Key \mapsto ClientCfg.Key])$

The SDP controller's exposure SPA service info .

VARIABLE $SDPSvrInfo$ @type: $[IP \mapsto SDPSvrCfg.IP, Port \mapsto SDPSvrCfg.Port]$

The number of replay attack messages inspected by SDP controller

VARIABLE *ReplayCount* @type: Integer;

The number of spoof attack messages inspected by *SDP* controller

VARIABLE *SpoofCount* @type: Integer;

The replay attack *Auth* sessions inspected by *SDP* controller in history recorded in *Log*.

VARIABLE *ReplaySession* @type: *uAuthSession*;

The spoof attack *Auth* sessions inspected by *SDP* controller in history recorded in *Log*.

VARIABLE *SpoofSession* @type: *uAuthSession*;

SDP controller's packets channnel for recieving control plane *Auth* messages, corresponds to its physical *NIC*.

VARIABLE *AuthChannel* @type: Sequence of SPA *Auth* Packets *Seq*([*MsgID* $\mapsto$ "SPA_AUTH",
sIP $\mapsto$ *uIP*,
sPort $\mapsto$ *RandomPort*(*uTstamp*, *USER_BASEPORT*),
dIP $\mapsto$ *uSDPSvrInfo.IP*, * The *SDP* Controller's *IP* and port for SPA protocol
dPort $\mapsto$ *uSDPSvrInfo.Port*,
ClientID $\mapsto$ *uID*, *Tstamp* $\mapsto$ *uTstamp*, * increased each session to anti Replay
SvrIP $\mapsto$ *Encrypt*(*uSvrInfo.IP*, *Key*), * Target Server's exposure service *Info*, need to kept secret
SvrPort $\mapsto$ *Encrypt*(*uSvrInfo.Port*, *Key*),
HMAC $\mapsto$ *CalcHMAC*(*uIP*, *uID*, *uTstamp*, *Encrypt*(*uSvrInfo.IP*, *Key*), *Encrypt*(*uSvrInfo.P*
, * *HMAC* of payload
Type $\mapsto$ *Set*("User", "Attacker")]); * Flag to indicate this message is built by legal user or attacker * this flag not invloved in inter-operation between *SDP* protocol entities,only for statistic;

The *SDP* controller's private variables (*AuthChannel* is public variable of *SDP* controller, for other entity can operat

sdpsvr_vars $\triangleq$ $\langle$*SDPSvrState*, *SDPSucSession*, *Account*, *SDPSvrInfo*, *ReplayCount*, *SpoofCount*,

The variables related to FireWall's state machine

The *FireWall*'s status indicates this entity's service is available or faulty.

The *FireWall* works in deny mode by default.

VARIABLE *FwState* @type: *Set*("Work")

Current *Acl Rule* Set maintained by the *FireWall* for data plane traffic.

VARIABLE *AclRuleSet* @type: *Set*([*sIP* $\mapsto$ *Integer*,
sPort $\mapsto$ Integer, * the value can be *MATCH_ANY*,
dIP $\mapsto$ Integer,

$dPort \mapsto$ Integer,
$protocol \mapsto$ "TCP",
$action \mapsto$ "Accept"])

The aged *Acl* Rules in history recorded in *FireWall*'s log.

VARIABLE *AgedRuleSet* @type: $Set([sIP \mapsto Integer,$
$sPort \mapsto$ Integer, * the value can be $MATCH\_ANY$,
$dIP \mapsto$ Integer,
$dPort \mapsto$ Integer,
$protocol \mapsto$ "TCP",
$action \mapsto$ "Accept"])

The dropped packets by *FireWall* in history recorded in log.

VARIABLE *DropPackets* @type: $Set([sIP \mapsto p.dIP,$ * *Only data plane TCP packets are processed by FireWall*
$sPort \mapsto p.dPort,$
$dIP \mapsto p.sIP,$
$dPort \mapsto p.sPort,$
$Flg \mapsto Set($"TCP_SYN", "TCP_SYN_ACK", "TCP_ACK"), * *TCP* handshake packets type.
$Type \mapsto Set($"User", "Attacker")];)

FireWall's control plane channnel for receiving *Openflow* instruction from *SDP* controller to configure data access *Acl Rule*, corresp

VARIABLE *FwCtlChannel* @type: Sequence of *Acl* config instructions $Seq([Rule \mapsto AclRule, op \mapsto Set($"Add", "*Del*")])$

FireWall's ingress data plane channnel for receiving packets from end point entities , corresponds to one of its physical *NIC*.

VARIABLE *FwDataChannel* @type: Sequence of Data Packets $Seq([sIP \mapsto p.dIP,$ * *Only data plane TCP packets are pr*
$sPort \mapsto p.dPort,$
$dIP \mapsto p.sIP,$
$dPort \mapsto p.sPort,$
$Flg \mapsto Set($"TCP_SYN", "TCP_SYN_ACK", "TCP_ACK"), * *TCP* handshake packets type.
$Type \mapsto Set($"User", "Attacker")];)

The *FireWall*'s private variables (*FwDataChannel* and *FwCtlChannel* are public variable of *FW*, for other entity can operate and

$fw\_vars \triangleq \langle FwState, AclRuleSet, AgedRuleSet, DropPackets \rangle$

The variables related to Attacker's state machine

The *Attacker*'s status indicates this entity's is spying or not.

VARIABLE *aState* @type: $Set($"Listen")

The *Attacker*'s current knowledge about legal user's auth action learned by sniffing legal user's auth message.

VARIABLE *AuthKnowledge* @type: *uAuthSession*

The *Attacker* initiated SPA attack sesssions in history recorded in log. Each session is identified by a fake SPA message.

VARIABLE *aSession* @type: *uAuthSession*

The *Attacker* initiated *TCP* connections towards the target server. Each link corresponds to an inspection attack to t

VARIABLE *aTCPLinkSet* @type:*Set*([*sIP* $\mapsto$ *Integer*,
sPort $\mapsto$ Integer,
dIP $\mapsto$ Integer,
dPort $\mapsto$ Integer,
State $\mapsto$ {"SYN_SENT", "ESTABLISHED"}
AuthID $\mapsto$ Integer] * The *AuthID* is used for relating to a captured auth message
) * For this model, once the attacker spy a SPA message, it will undertake a data attack to the target server.
* The value *UNKNOWN_AUTH_ID* indicates the attack is not originate from a captured auth message, but a captured data message

The number of sucessfully sniffed SPA messages by attacker

VARIABLE *sniffCount* @type: Integer;

All the successfully sniffed SPA messages by attacker in history recorded in log

VARIABLE *CapAuthMsg* @type: *uAuthSession*;

Attacker maintained increasing sequence number to build local port field for *TCP* links of different *Dtection* attack.

VARIABLE *aCounter* @type: Integer;

Attacker's *IP* address, which is got by configuration.

If *NAT_FLAG* = TRUE, then attacker and legal user located in the same *LAN* and share same public *IP* (*aIP* = *uIP*).

VARIABLE *aIP* @type: Integer;

The *Attacker*'s current knowledge about legal user's data access learned by sniffing legal user's *TCP* handshake packe

VARIABLE *DataKnowledge* @type: *Set*([*sIP* $\mapsto$ *p.dIP*, * *Only data plane TCP packets are processed by F*
sPort $\mapsto$ *p.dPort*,
dIP $\mapsto$ *p.sIP*,
dPort $\mapsto$ *p.sPort*,
Flg $\mapsto$ *Set*("TCP_SYN", "TCP_SYN_ACK", "TCP_ACK"), * *TCP* handshake packets type.
Type $\mapsto$ *Set*("User", "Attacker")])

All the successfully sniffed user data packets by attacker in history recorded in log

VARIABLE *CapDataMsg* @type: *DataKnowledge*

The attacker's packets channnel for recieving data plane packets, corresponds to its physical *NIC*.

VARIABLE *aChannel* @type: *uChannel*

The attacker's private variables (*aChannel* is public variable of attacker, for other entity can operate and modify *aChannel* variable

$attacker\_vars \triangleq \langle aState, AuthKnowledge, aSession, aTCPLinkSet, sniffCount, CapAuthMsg, aCounter, aII$

The variables related to target service server's state machine

The target server's status indicates this entity's service is available or faulty.

VARIABLE $sState$ @type: $Set(\text{"Listen"})$

The TCP socket maintained in server side initiated from end points towards target server.

VARIABLE $sTCPLinkSet$ @type: $Set([sIP \mapsto p.dIP, \backslash * Only\ data\ plane\ TCP\ packets\ are\ processed\ by\ FireWall$
$sPort \mapsto p.dPort,$
$dIP \mapsto p.sIP,$
$dPort \mapsto p.sPort,$
$Flg \mapsto Set(\text{"TCP\_SYN"}, \text{"TCP\_SYN\_ACK"}, \text{"TCP\_ACK"}), \backslash * TCP$ handshake packets type.
$Type \mapsto Set(\text{"User"}, \text{"Attacker"})])$

The target server's exposure service info got from configuration.

VARIABLE $sSvrInfo$ @type: $[IP \mapsto SvrCfg.IP, Port \mapsto SvrCfg.Port]$

The server's packets channnel for recieving data plane packets from endpoint equipments, corresponds to its physical NIC.

VARIABLE $sChannel$ @type: $uChannel$

The target server's private variables (*sChannel* is public variable of server, for other entity can operate and modify *sChannel* variab

$server\_vars \triangleq \langle sState, sTCPLinkSet, sSvrInfo \rangle$

All the public variables of the model

$uChannel$:$Intf1$, $aChannel$: $Intf2$, $AuthChannel$: $Intf3$, $FwCtlChannel$: $Intf4$, $FwDataChannel$: $Intf5$, $sChannel$: $Intf6$

$Public\_vars \triangleq \langle uChannel, AuthChannel, FwCtlChannel, FwDataChannel, aChannel, sChannel \rangle$

All the variables that consititute the global state machine

$vars \triangleq \langle user\_vars, sdpsvr\_vars, fw\_vars, attacker\_vars, server\_vars, Public\_vars \rangle$

Common functions and operators

Sequence to Set

RECURSIVE $Seq2Set(\_)$

$Seq2Set(S) \triangleq$

IF $S = \langle\rangle$ THEN $\{\}$

ELSE

$$\begin{array}{l}\text{LET } i \triangleq Head(S)\\ \text{IN } \{i\} \cup Seq2Set(Tail(S))\end{array}$$

Select local port randomly when client create socket connection

$$RandomPort(count,\ base) \triangleq (\text{CHOOSE } x \in (count + base)\ ..\ (base + 100) : \text{TRUE})$$

simulate Symmetric-key algorithm: *Encrypt* function, this operator simplified by *XOR* operation

$$\begin{array}{l}Encrypt(d,\ k) \triangleq\\ \quad \text{LET RECURSIVE } XorPureR(\_,\ \_,\ \_,\ \_)\\ \qquad XorPureR(x,\ y,\ n,\ m) \triangleq\\ \qquad\quad \text{IF } m = 0\\ \qquad\quad \text{THEN } 0\\ \qquad\quad \text{ELSE LET } exp \triangleq 2^n\\ \qquad\qquad\quad \text{IN } exp * (((x \div exp) + (y \div exp))\%2)\\ \qquad\qquad\qquad\quad + XorPureR(x,\ y,\ n+1,\ m \div 2)\\ \quad \text{IN IF } d \geq k \text{ THEN } XorPureR(d,\ k,\ 0,\ d) \text{ ELSE } XorPureR(k,\ d,\ 0,\ k)\end{array}$$

simulate Symmetric-key algorithm: Decrypt function

$$DeCrypt(d,\ k) \triangleq Encrypt(d,\ k)$$

simulate *HMAC* function for SPA message

$$CalcHMAC(n1,\ n2,\ n3,\ n4,\ n5,\ key) \triangleq Encrypt(n1 + n2 + n3 + n4 + n5,\ key)$$

Init state description of legal user

User *Init*: Read configuration and ready to launch an access to target server

the init state is ready to start a auth session.

$$\begin{array}{rl}UsrInit \triangleq & \wedge\ uState = \text{“Start\_Auth”}\\ & \wedge\ uID = \ ClientCfg.LoginID\\ & \wedge\ Key = ClientCfg.Key\\ & \wedge\ uIP = ClientCfg.SrcIp\\ & \wedge\ uTstamp = 0\\ & \wedge\ uSDPSvrInfo = [IP \mapsto SDPSvrCfg.IP,\ Port \mapsto SDPSvrCfg.Port]\\ & \wedge\ uSvrInfo = [IP \mapsto SvrCfg.IP,\ Port \mapsto SvrCfg.Port]\\ & \wedge\ uTCPLinkSet = \{\}\\ & \wedge\ uChannel = \langle\rangle\end{array}$$

$\wedge\ uAuthSession = \{\}$

Next state actions of legal user

Action 1: *UsrCommitSpaAuth*

Legal user perform SPA (Single Packet Authentication) session by sending a SPA packet to *SDP* controller.

Variables changed: $< uState, uAuthSession, uTstamp, AuthChannel >$

$UsrCommitSpaAuth \triangleq$

$\wedge\ uState = \text{“Start\_Auth”}$

$\wedge\ uState' = \text{“Auth\_End”}$

$\wedge\ uTstamp' = uTstamp + 1$ $uTstamp$ increases each session for anti-replay.

$\wedge\ AuthChannel' = Append(AuthChannel,$

$[MsgID \mapsto \text{“SPA\_AUTH”},$

$sIP \mapsto uIP,$

$sPort \mapsto RandomPort(uTstamp,\ USER\_BASEPORT),$

$dIP \mapsto uSDPSvrInfo.IP,$

$dPort \mapsto uSDPSvrInfo.Port,$

$ClientID \mapsto uID,$

$Tstamp \mapsto uTstamp,$

$SvrIP \mapsto Encrypt(uSvrInfo.IP,\ Key),$

$SvrPort \mapsto Encrypt(uSvrInfo.Port,\ Key),$

$HMAC \mapsto CalcHMAC(uIP,\ uID,\ uTstamp,\ Encrypt(uSvrInfo.IP,\ Key),$
$Encrypt(uSvrInfo.Port,\ Key),\ Key),$

$Type \mapsto \text{“User”}]$

)

$\wedge\ uAuthSession' = uAuthSession \cup \{Head(AuthChannel')\}$ Auth session is recorded in *Log*

$\wedge$ UNCHANGED $\langle uIP,\ uID,\ Key,\ uSDPSvrInfo,\ uSvrInfo,\ uTCPLinkSet\rangle$

$\wedge$ UNCHANGED $sdpsvr\_vars$

$\wedge$ UNCHANGED fw_vars

$\wedge$ UNCHANGED $attacker\_vars$

$\wedge$ UNCHANGED $server\_vars$

$\wedge$ UNCHANGED $\langle uChannel,\ FwCtlChannel,\ FwDataChannel,\ aChannel,\ sChannel\rangle$

$\wedge$ UNCHANGED $\langle vars \setminus (uState,\ uTstamp,\ AuthChannel,\ uAuthSession)\rangle$

Action 2: *UsrConnectSvr*

Legal user try to access target service server after perform SPA (Single Packet Authentication) session.

the first action to connect the server is sending *TCP SYN* packets.

Variables changed: $< uState, uTCPLinkSet, FwDataChannel >$

$LatestAuthSession \triangleq \text{CHOOSE } x \in uAuthSession : (\forall\, y \in uAuthSession : x.Tstamp \geq y.Tstamp)$

$UsrBulidTcpSynPkt \triangleq$

$[sIP \mapsto uIP,$
$sPort \mapsto LatestAuthSession.sPort + 1,$ the new data access corresponds to the latest auth session by
$dIP \mapsto uSvrInfo.IP,$
$dPort \mapsto uSvrInfo.Port,$
$Flg \mapsto$ "TCP_SYN",
$Type \mapsto$ "User"$]$

$UsrConnectSvr \triangleq$

$\wedge\ uState = $ "Auth_End"
$\wedge\ uState' = $ "Connecting" the user now waiting for *TCP* handshakes over.
$\wedge\ uAuthSession \neq \{\}$
$\wedge\ uTCPLinkSet = \{\}$
$\wedge\ uTCPLinkSet' = \{$ We assume the user only launch one data access session.
$[sIP \mapsto UsrBulidTcpSynPkt.sIP,$
$sPort \mapsto UsrBulidTcpSynPkt.sPort,$
$dIP \mapsto UsrBulidTcpSynPkt.dIP,$
$dPort \mapsto UsrBulidTcpSynPkt.dPort,$
$State \mapsto$ "SYN_SENT" Create new *TCP* socket corresponds to the latest *Auth* session, *TCP* link
$] \}$
$\wedge\ FwDataChannel' = Append(FwDataChannel, UsrBulidTcpSynPkt)$ Send *TCP SYN* packet to *F*
$\wedge$ UNCHANGED $\langle uIP, uID, Key, uTstamp, uSDPSvrInfo, uSvrInfo, uAuthSession\rangle$
$\wedge$ UNCHANGED $sdpsvr\_vars$
$\wedge$ UNCHANGED fw_vars
$\wedge$ UNCHANGED $attacker\_vars$
$\wedge$ UNCHANGED $server\_vars$
$\wedge$ UNCHANGED $\langle uChannel, AuthChannel, FwCtlChannel, aChannel, sChannel\rangle$

Action 3: *UsrRcvSynAck*

Legal user receive $TCP\ SYN\ Ack$ packet from target server which
indicates data TCP link establised. This represents the user has
successfully fufilled a data access.
Variables changed: $< uState,\ uTCPLinkSet,\ uChannel, FwDataChannel >$

$$HasMatchLink(p,\ LinkSet) \triangleq$$
$$\exists\, x \in LinkSet: \ \begin{aligned} &\wedge\ p.sIP = x.dIP \\ &\wedge\ p.sPort = x.dPort \\ &\wedge\ p.dIP = x.sIP \\ &\wedge\ p.dPort = x.sPort \end{aligned}$$

$GetMatchLink(p,\ LinkSet) \triangleq$ get match TCB (TCP control Block) for a received TCP packet

$$\text{CHOOSE } x \in LinkSet: \ \begin{aligned} &\wedge\ p.sIP = x.dIP \\ &\wedge\ p.sPort = x.dPort \\ &\wedge\ p.dIP = x.sIP \\ &\wedge\ p.dPort = x.sPort \end{aligned}$$

$EndPointBulidTcpAckPkt(p,\ t) \triangleq$ End point equipments might be a legal user or attacker

$$\begin{aligned} [&sIP \quad \mapsto p.dIP, \\ &sPort \quad \mapsto p.dPort, \\ &dIP \quad \mapsto p.sIP, \\ &dPort \quad \mapsto p.sPort, \\ &Flg \quad \mapsto \text{“TCP\_ACK”}, \\ &Type \quad \mapsto t] \end{aligned}$$

$UsrRcvSynAck \triangleq$

$$\begin{aligned} &\wedge\ uState = \text{“Connecting”} \\ &\wedge\ uTCPLinkSet \neq \{\} \\ &\wedge\ uChannel \neq \langle\rangle \\ &\wedge\ Head(uChannel).Flg \ = \text{“TCP\_SYN\_ACK”} \\ &\wedge\ Head(uChannel).Type = \text{“User”} \\ &\wedge\ HasMatchLink(Head(uChannel),\ uTCPLinkSet) \text{ Receive } TCP\_SYN\_ACK \text{ from target server and match the conne} \\ &\wedge\ uTCPLinkSet' = (uTCPLinkSet \setminus \{GetMatchLink(Head(uChannel),\ uTCPLinkSet)\}) \\ &\qquad\qquad \cup \{[sIP \quad \mapsto GetMatchLink(Head(uChannel),\ uTCPLinkSet).sIP, \end{aligned}$$

$sPort \mapsto GetMatchLink(Head(uChannel),\ uTCPLinkSet).sPort,$
$dIP \mapsto GetMatchLink(Head(uChannel),\ uTCPLinkSet).dIP,$
$dPort \mapsto GetMatchLink(Head(uChannel),\ uTCPLinkSet).dPort,$
$State \mapsto$ "ESTABLISHED" Updata *TCP* link status to established
]
}
$\wedge\ uState' =$ "Connected" The user successfully access the target server
$\wedge\ uChannel' = Tail(uChannel)$ Send *TCP ACK* packet (the last step of hand shake) to target server
$\wedge\ FwDataChannel' = Append(FwDataChannel,\ EndPointBulidTcpAckPkt(Head(uChannel),$ "
$\wedge$ UNCHANGED $\langle uIP,\ uID,\ Key,\ uTstamp,\ uSDPSvrInfo,\ uSvrInfo,\ uAuthSession\rangle$
$\wedge$ UNCHANGED $sdpsvr\_vars$
$\wedge$ UNCHANGED fw_vars
$\wedge$ UNCHANGED $attacker\_vars$
$\wedge$ UNCHANGED $server\_vars$
$\wedge$ UNCHANGED $\langle AuthChannel,\ FwCtlChannel,\ aChannel,\ sChannel\rangle$

Init state description of SDP Controller

SDP Controller *Init*: Read configuration and ready to provide SPA auth service.

$SDPSvrInit \triangleq$ $\wedge\ SDPSvrState =$ "Work"
$\wedge\ SDPSucSession = \{\}$
$\wedge\ Account = \{[ClientID \mapsto ClientCfg.LoginID,\ Key \mapsto ClientCfg.Key]\}$ Load user
$\wedge\ SDPSvrInfo = [IP \mapsto SDPSvrCfg.IP,\ Port \mapsto SDPSvrCfg.Port]$ Service *IP* and p
$\wedge\ AuthChannel = \langle\rangle$
$\wedge\ ReplayCount = 0$
$\wedge\ SpoofCount = 0$
$\wedge\ ReplaySession = \{\}$
$\wedge\ SpoofSession = \{\}$

Next state actions of SDP Controller

Action 4: *SDPSvrProcSpaAuth*

SDP Controller process received SPA message.

Scenario 3: Request from legal user, controller then instruct firewall to admit data access after authenticaiton.

Scenario 1 2: controller recognize spoof and replay attack.

Variables changed: $< AuthChannel, SDPSucSession, ReplaySession, SpoofSession, ReplayCount, SpoofCount, FwCtlChannel >$

if a coming SPA message SN match the history message recorded in anti-replay window

then it must be recognized as a replay attack packet.

$FindAntiReplay(msg, wnd) \triangleq$

IF $\exists\, r \in wnd : (msg.ClientID = r.ClientID \land msg.Tstamp = r.Tstamp)$

THEN

TRUE

ELSE

FALSE

For a recognized replay attack message, SDP controller drop it and recorded in the log.

$SDPSvrAntiReplayAtk \triangleq$

$\land\ AuthChannel' = Tail(AuthChannel)$ Drop packet

$\land\ ReplayCount' = ReplayCount + 1$ Increase statistics

$\land\ ReplaySession' = ReplaySession \cup \{Head(AuthChannel)\}$ Update log

For a recognized spoof attack message, SDP controller drop it and recorded in the log.

$SDPSvrAntiSpoof \triangleq$

$\land\ AuthChannel' = Tail(AuthChannel)$ Drop packet

$\land\ SpoofCount' = SpoofCount + 1$ Increase statistics

$\land\ SpoofSession' = SpoofSession \cup \{Head(AuthChannel)\}$ Update log

SDP controller implement authenticaiton triggered by a received SPA message

The authentication is implemented by recaculate the $HMAC$ according the user account $Info$

$SpaProcAuth(msg, accounts) \triangleq$

$\exists\, a \in accounts : (\ \land\ a.ClientID = msg.ClientID$ user ID must match

Recaclulate the $HMAC$ value by using local stored user Key and then compare the value of corresponding field in SPA packet.

$\land\ CalcHMAC(msg.sIP,\ msg.ClientID,\ msg.Tstamp,\ msg.SvrIP,\ msg.SvrPort,\ a.Key) =$

)

Get the correspond key by user ID from IAM stored accounts

$GetKey(id, accounts) \triangleq$ (CHOOSE $a \in accounts : a.ClientID = id).Key$

SDP controller instruct $FireWall$ to config $Acl\ Rule$ by sending instruction message to $FireWall$'s control plane channel

$SDPSvrCfgFw(Acl,\ op)\ \triangleq$

$\land\ FwCtlChannel' = Append(FwCtlChannel,\ [Rule \mapsto Acl,\ op \mapsto op])$

$SDPSvrProcSpaAuth\ \triangleq$

$\land\ SDPSvrState = \text{“Work”}$

$\land\ AuthChannel \neq \langle\rangle$

$\land\ Head(AuthChannel).MsgID = \text{“SPA\_AUTH”}$ check the packet is SPA message

$\land\ Head(AuthChannel).dIP = SDPSvrInfo.IP$

$\land\ Head(AuthChannel).dPort = SDPSvrInfo.Port$

$\land$ IF $FindAntiReplay(Head(AuthChannel),\ SDPSucSession) =$ TRUE case 1: the packet is a replay r

THEN

$\land\ SDPSvrAntiReplayAtk$ drop packets and record exception into log

$\land$ UNCHANGED $user\_vars$

$\land$ UNCHANGED $\langle SDPSvrState,\ SDPSucSession,\ Account,\ SDPSvrInfo,\ SpoofCount,\ SpoofS$

$\land$ UNCHANGED fw_vars

$\land$ UNCHANGED $attacker\_vars$

$\land$ UNCHANGED $server\_vars$

$\land$ UNCHANGED $\langle uChannel,\ FwCtlChannel,\ FwDataChannel,\ aChannel,\ sChannel\rangle$

ELSE

$\land$ IF $SpaProcAuth(Head(AuthChannel),\ Account) =$ FALSE case 2: it is a spoof message or from

THEN

$\land\ SDPSvrAntiSpoof$ drop packets and record exception into log

$\land$ UNCHANGED $user\_vars$

$\land$ UNCHANGED $\langle SDPSvrState,\ SDPSucSession,\ Account,\ SDPSvrInfo,\ ReplayCount,\ A$

$\land$ UNCHANGED fw_vars

$\land$ UNCHANGED $attacker\_vars$

$\land$ UNCHANGED $server\_vars$

$\land$ UNCHANGED $\langle uChannel,\ FwCtlChannel,\ FwDataChannel,\ aChannel,\ sChannel\rangle$

ELSE case 3: Authenticated successfully, then send instruction to FW to allow data access towards ta

$\land\ SDPSvrCfgFw([sIP \mapsto Head(AuthChannel).sIP,$

$sPort \mapsto MATCH\_ANY,$ this *Acl Rule* is 3 tuple, for data access sourc

$dIP \mapsto DeCrypt(Head(AuthChannel).SvrIP,\ GetKey(Head(Aut$

$dPort \mapsto DeCrypt(Head(AuthChannel).SvrPort,\ GetKey(Head(A$

$protocol \mapsto$ "TCP",
$action \mapsto$ "Accept"],
"Add" The instruction code is to *Add* a new rule.
)
$\wedge\ SDPSucSession' = SDPSucSession \cup \{Head(AuthChannel)\}$ record in log
$\wedge\ AuthChannel' = Tail(AuthChannel)$
$\wedge$ UNCHANGED $user\_vars$
$\wedge$ UNCHANGED $\langle SDPSvrState, Account, SDPSvrInfo, ReplayCount, SpoofCount, ReplaySession, S$
$\wedge$ UNCHANGED fw_vars
$\wedge$ UNCHANGED $attacker\_vars$
$\wedge$ UNCHANGED $server\_vars$
$\wedge$ UNCHANGED $\langle uChannel, FwDataChannel, aChannel, sChannel\rangle$

Init state description of FireWall

Fire wall init: power on and enter work state, by default, it works in deny mode and will drop any ingress data packets.

$FwInit \triangleq$
$\wedge\ FwCtlChannel = \langle\rangle$
$\wedge\ FwDataChannel = \langle\rangle$
$\wedge\ FwState =$ "Work"
$\wedge\ AclRuleSet = \{\}$
$\wedge\ AgedRuleSet = \{\}$
$\wedge\ DropPackets = \{\}$

Next state actions of FireWall

Action 5: *FwProcAclCfg*

FireWall receive *Acl Rule* config instruction from control plane channel, and hence create a 3 Tuple rule for data access

Variables changed: $< FwCtlChannel, AclRuleSet >$

$FwProcAclCfg \triangleq$
$\wedge\ FwState =$ "Work"
$\wedge\ FwCtlChannel \neq \langle\rangle$
$\wedge\ Head(FwCtlChannel).op =$ "Add" Check instruction message format
$\wedge\ AclRuleSet' = AclRuleSet \cup \{Head(FwCtlChannel).Rule\}$ Update local maintained rule table
$\wedge\ FwCtlChannel' = Tail(FwCtlChannel)$

$\wedge$ UNCHANGED $user\_vars$

$\wedge$ UNCHANGED $sdpsvr\_vars$

$\wedge$ UNCHANGED $attacker\_vars$

$\wedge$ UNCHANGED $server\_vars$

$\wedge$ UNCHANGED $\langle FwState,\ AgedRuleSet,\ DropPackets\rangle$

$\wedge$ UNCHANGED $\langle uChannel,\ AuthChannel,\ FwDataChannel,\ aChannel,\ sChannel\rangle$

Action 6: $FwProcEndPointAccess$

$FireWall$ receive a ingress data packet from end point side and implement filtering function according to $L3\ ACL\ Rul$

Variables changed: $< sChannel,\ AclRuleSet, FwDataChannel, DropPackets >$

Whether the TCP packet match a given 3 tuple rule.

$$
\begin{aligned}
&AclMatch3Tuple(p,\ Acl)\ \triangleq \\
&\quad \exists\, r \in Acl : (\ \wedge\ p.sIP = r.sIP \\
&\qquad\qquad\ \wedge\ p.dIP = r.dIP \\
&\qquad\qquad\ \wedge\ r.sPort = MATCH\_ANY \quad \text{don't care source port value} \\
&\qquad\qquad\ \wedge\ p.dPort = r.dPort \\
&\qquad\qquad\ \wedge\ r.action = \text{“Accept”}\ )
\end{aligned}
$$

Whether the TCP packet match a given 4 tuple rule.

$$
\begin{aligned}
&AclMatch4Tuple(p,\ Acl)\ \triangleq \\
&\quad \exists\, r \in Acl : (\ \wedge\ p.sIP = r.sIP \quad (sIP,sPort,dIP,dPort) \text{ must match exactly} \\
&\qquad\qquad\ \wedge\ p.dIP = r.dIP \\
&\qquad\qquad\ \wedge\ r.sPort \neq MATCH\_ANY \\
&\qquad\qquad\ \wedge\ r.sPort = p.sPort \\
&\qquad\qquad\ \wedge\ p.dPort = r.dPort \\
&\qquad\qquad\ \wedge\ r.action = \text{“Accept”}\ )
\end{aligned}
$$

The firewall automatically create an exactly matched 4 tuple rule according to a received new TCP link packets

The 3 tuple rule configed by SDP controller by default with RELATED attribute, which means a new TCP link packe

can trigger creating of a exactly matched 4 tuple rule.

$$
\begin{aligned}
&CreateRelatedRule(p)\ \triangleq \\
&\quad [sIP \quad\ \mapsto p.sIP, \\
&\quad\ sPort \quad \mapsto p.sPort,
\end{aligned}
$$

$dIP \quad \mapsto p.dIP,$
$dPort \quad \mapsto p.dPort,$
$protocol \mapsto$ "TCP",
$action \quad \mapsto$ "Accept"]

$FwProcEndPointAccess \triangleq$
$\wedge\ FwState =$ "Work"
$\wedge\ FwDataChannel \neq \langle\rangle$
$\wedge$ ($\vee\ Head(FwDataChannel).Flg =$ "TCP_SYN" to simplify the model, we only consider TCP connection proceudre f
$\vee\ Head(FwDataChannel).Flg =$ "TCP_ACK" the end point euipments as TCP client, only send TCP_SYN and T
)
$\wedge$ (IF $AclMatch4Tuple(Head(FwDataChannel),\ AclRuleSet)$
THEN $CASE1$: the incoming packets exactly match a 4 tuple rule
$\wedge\ sChannel' = Append(sChannel,\ Head(FwDataChannel))$ route the packets to target server
$\wedge\ FwDataChannel' = Tail(FwDataChannel)$
$\wedge\ AclRuleSet' = AclRuleSet$
$\wedge\ DropPackets' = DropPackets$
ELSE
(IF $AclMatch3Tuple(Head(FwDataChannel),\ AclRuleSet)$
THEN $CASE2$: the incoming packets only match a 3 tuple rule
$\wedge\ sChannel' = Append(sChannel,\ Head(FwDataChannel))$ route the packets to target server
$\wedge\ AclRuleSet' = AclRuleSet \cup \{CreateRelatedRule(Head(FwDataChannel))\}$
$\wedge\ FwDataChannel' = Tail(FwDataChannel)$ This is a new TCP link, so create a exactly matched 4 tuple rule ar
$\wedge\ DropPackets' = DropPackets$
ELSE $CASE3$: the incoming packetsnot match any rule
$\wedge\ FwDataChannel' = Tail(FwDataChannel)$
$\wedge\ AclRuleSet' = AclRuleSet$
$\wedge\ sChannel' = sChannel$ just drop the packets
$\wedge\ DropPackets' = DropPackets \cup \{Head(FwDataChannel)\}$ record it into exception log
)
)
$\wedge$ UNCHANGED $user\_vars$
$\wedge$ UNCHANGED $sdpsvr\_vars$

$\land$ UNCHANGED $attacker\_vars$

$\land$ UNCHANGED $\langle FwState,\ AgedRuleSet\rangle$

$\land$ UNCHANGED $server\_vars$

$\land$ UNCHANGED $\langle uChannel,\ AuthChannel,\ FwCtlChannel,\ aChannel\rangle$

Action 7: $FwProcAclTimeOut$

A 3 Tuple Acl rule configed by SDP controller automatically deleted due to aging mechanism.

Variables changed: $< AclRuleSet, AgedRuleSet >$

$FwProcAclTimeOut \triangleq$

$\land\ FwState =$ "Work"

$\land\ AclRuleSet \neq \{\}$

$\land \exists\, r \in AclRuleSet : r.sPort = MATCH\_ANY$ only 3 tuple rule with aging mechanism

$\land\ AclRuleSet' = AclRuleSet \setminus \{\text{CHOOSE } r \in AclRuleSet : r.sPort = MATCH\_ANY\}$ aging and

$\land\ AgedRuleSet' = AgedRuleSet \cup \{\text{CHOOSE } r \in AclRuleSet : r.sPort = MATCH\_ANY\}$ record

$\land$ UNCHANGED $user\_vars$

$\land$ UNCHANGED $sdpsvr\_vars$

$\land$ UNCHANGED $attacker\_vars$

$\land$ UNCHANGED $\langle FwState,\ DropPackets\rangle$

$\land$ UNCHANGED $server\_vars$

$\land$ UNCHANGED $Public\_vars$

Init state description of target service server

Target TCP server init and begin listening on its service IP and $Port$.

$ServerInit \triangleq$ $\land\ sState =$ "Listen"

$\land\ sSvrInfo = [IP \mapsto SvrCfg.IP,\ Port \mapsto SvrCfg.Port]$ Load configuration

$\land\ sTCPLinkSet = \{\}$

$\land\ sChannel = \langle\rangle$

Next state actions of target service server

Action 8: $ServerRcvTCPSyn$

Target server recieve a $TCP\ SYN$ packet from client side and try to allocate a new TCB.

Because the Firewall dose not filter server to endpoint direction packets, so to simplify the model, the server direcly se

$uChannel$.

Variables changed: $< sTCPLinkSet, sChannel, uChannel, aChannel >$

Whether the coming packet indicates a new connection

$NewLink(p,\ LinkSet) \triangleq$
IF $LinkSet = \{\}$
THEN
TRUE
ELSE
(IF $\forall x \in LinkSet : ($ without matching TCB (TCP Control Block)
$\lor x.sIP \neq p.sIP$
$\lor x.dIP \neq p.dIP$
$\lor x.sPort \neq p.sPort$
$\lor x.dPort \neq p.dPort)$
THEN
TRUE
ELSE
FALSE)

$ServerRcvTCPSyn \triangleq$
$\land\ sState =$ "Listen"
$\land\ sChannel \neq \langle\rangle$
$\land\ Head(sChannel).Flg\ =$ "TCP_SYN"
$\land\ Head(sChannel).dIP = sSvrInfo.IP$ check incoming packets format
$\land\ Head(sChannel).dPort = sSvrInfo.Port$
$\land\ sChannel' = Tail(sChannel)$
$\land$ (IF $NewLink(Head(sChannel),\ sTCPLinkSet)$
THEN $CASE1$: New $TCP\ SYN$ packets
$\land\ sTCPLinkSet' = sTCPLinkSet \cup \{$ create a TCB and update local link set.
$[dIP \mapsto Head(sChannel).sIP,$
$dPort \mapsto Head(sChannel).sPort,$
$sIP \mapsto Head(sChannel).dIP,$
$sPort \mapsto Head(sChannel).dPort,$
$Type \mapsto Head(sChannel).Type,$

$State \quad \mapsto$ "SYN_RCVD" the TCB 's state is SYN_RCVD

]}

$\wedge$ (IF $Head(sChannel).Type =$ "User"

THEN If the client is legal user, then send TCP_SYN_ACK packet to legal user.

($\wedge\ uChannel' = Append(uChannel, [$

$sIP \quad \mapsto Head(sChannel).dIP,$

$sPort \quad \mapsto Head(sChannel).dPort,$

$dIP \quad \mapsto Head(sChannel).sIP,$

$dPort \quad \mapsto Head(sChannel).sPort,$

$Flg \quad \mapsto$ "TCP_SYN_ACK",

$Type \quad \mapsto Head(sChannel).Type]$

)

$\wedge\ aChannel' = aChannel$

)

ELSE If the client is attacker, then send TCP_SYN_ACK packet to attacker.

($\wedge\ aChannel' = Append(aChannel, [$

$sIP \quad \mapsto Head(sChannel).dIP,$

$sPort \quad \mapsto Head(sChannel).dPort,$

$dIP \quad \mapsto Head(sChannel).sIP,$

$dPort \quad \mapsto Head(sChannel).sPort,$

$Flg \quad \mapsto$ "TCP_SYN_ACK",

$Type \quad \mapsto Head(sChannel).Type]$

)

$\wedge\ uChannel' = uChannel$

)

)

ELSE $CASE2$: duplicated $TCP\ SYN$ packet,just neglect it for we don't focus on $TCP\ SYN$ Flood attack.

$\wedge \quad sTCPLinkSet' = sTCPLinkSet$

$\wedge \quad aChannel' = aChannel$

$\wedge \quad uChannel' = uChannel$

)

$\wedge$ UNCHANGED $user\_vars$

$\wedge$ UNCHANGED $sdpsvr\_vars$

$\land$ UNCHANGED $attacker\_vars$

$\land$ UNCHANGED $\langle sState,\ sSvrInfo \rangle$

$\land$ UNCHANGED fw_vars

$\land$ UNCHANGED $\langle AuthChannel,\ FwCtlChannel,\ FwDataChannel \rangle$

Action 9: $ServerRcvTCPSyn$

Target server recieve a $TCP\ ACK$ packet that acknowledge the last SYN_ACK, then establish the TCP link with the client.

Variables changed: $< sTCPLinkSet, sChannel >$

$ServerRcvTcpAck \triangleq$

$\land\ sState =$ "Listen"

$\land\ sChannel \neq \langle\rangle$

$\land\ Head(sChannel).Flg =$ "TCP_ACK" check incoming packets format

$\land\ HasMatchLink(Head(sChannel),\ sTCPLinkSet)$

$\land\ GetMatchLink(Head(sChannel),\ sTCPLinkSet).State =$ "SYN_RCVD" the matched TCB state must be SYN

$\land\ sChannel' = Tail(sChannel)$

$\land\ sTCPLinkSet' = (sTCPLinkSet \setminus \{GetMatchLink(Head(sChannel),\ sTCPLinkSet)\})$

$$\cup \{[sIP \mapsto GetMatchLink(Head(sChannel),\ sTCPLinkSet).sIP,$$
$$sPort \mapsto GetMatchLink(Head(sChannel),\ sTCPLinkSet).sPort,$$
$$dIP \mapsto GetMatchLink(Head(sChannel),\ sTCPLinkSet).dIP,$$
$$dPort \mapsto GetMatchLink(Head(sChannel),\ sTCPLinkSet).dPort,$$
$$Type \mapsto GetMatchLink(Head(sChannel),\ sTCPLinkSet).Type,$$
$$State \mapsto \text{"ESTABLISHED"}$$

Update TCP link state to $ESTABLISHED$.

This indicates the client has successfully accessed target server

$$]\}$$

$\land$ UNCHANGED $user\_vars$

$\land$ UNCHANGED $sdpsvr\_vars$

$\land$ UNCHANGED $attacker\_vars$

$\land$ UNCHANGED $\langle sState,\ sSvrInfo \rangle$

$\land$ UNCHANGED fw_vars

$\land$ UNCHANGED $\langle uChannel,\ AuthChannel,\ FwCtlChannel,\ FwDataChannel,\ aChannel \rangle$

Init state description of Attacker

Attacker init and capable of sniffing the packets on the local network.

$AttackerInit \triangleq\ \land\ aState =$ "Listen"

$\wedge\ AuthKnowledge = \{\}$
$\wedge\ aSession = \{\}$
$\wedge\ aTCPLinkSet = \{\}$
$\wedge\ aChannel = \langle\rangle$
$\wedge\ sniffCount \quad = 0$
$\wedge\ CapAuthMsg = \{\}$
$\wedge\ aCounter = 0$
$\wedge\ aIP = AttackerCfg.SrcIp$
$\wedge\ DataKnowledge = \{\}$
$\wedge\ CapDataMsg = \{\}$

Next state actions of attacker

Action 10: *AttackerSniffAuthChannel*

Attacker eavesdropping SPA message from legal user to *SDP* controller by sniffing the *Auth* channel.

Once a new SPA message is captured,attacker will duplicate it into its current Auth-knowledge set.

We don't guarantee every new SPA message can be captured by attacker, it only has the opportuity to get each messag

Variables changed: $< AuthKnowledge, CapAuthMsg, sniffCount >$

Slect a new (which means unknown to attacker till now) SPA message from the *Auth* channel
to simulate a successful sniff.

$SelectNewAuthMsg(MsgQ,\ known) \triangleq$
IF $known \neq \{\}$
THEN for a dedicate user, the difference among SPA messages is the value of *SN* (counter) field.
CHOOSE $S \in$ SUBSET $Seq2Set(MsgQ) : (\forall\, x \in S : (\forall\, y \in known : x.Tstamp \neq y.Tstamp))$
ELSE
$Seq2Set(MsgQ)$

For the attacker can also insert fake messages into channel, but
for both data and auth channel, attacker only wants to capture messages from legal user.
so the *PureChannel*() function is to select the set of user's messages.

$PureChannel(S) \triangleq SelectSeq(S,$ LAMBDA $x : x.Type =$ "User")

$AttackerSniffAuthChannel \triangleq$
$\wedge\ aState =$ "Listen"

$\land\ PureChannel(AuthChannel) \neq \langle\rangle$ pre-condition: there exists attacker unknown legal user originated SPA messages on t

$\land\ \exists\, i \in 1\,..\,Len(PureChannel(AuthChannel)) :$

$\quad (\forall\, x \in CapAuthMsg : PureChannel(AuthChannel)[i].Tstamp \neq x.Tstamp)$

$\land\ AuthKnowledge' = AuthKnowledge \cup$ post-condition: attacker learned new intelligence by a successful sniffing.

$\quad SelectNewAuthMsg(PureChannel(AuthChannel),\ CapAuthMsg)$

$\land\ CapAuthMsg' = CapAuthMsg \cup$ All the captured message in history recorded in *Log*.

$\quad SelectNewAuthMsg(PureChannel(AuthChannel),\ CapAuthMsg)$

$\land\ sniffCount' = sniffCount + 1$ increase statistics

$\land$ UNCHANGED $user\_vars$

$\land$ UNCHANGED $sdpsvr\_vars$

$\land$ UNCHANGED fw_vars

$\land$ UNCHANGED $server\_vars$

$\land$ UNCHANGED $\langle aState,\ aSession,\ aTCPLinkSet,\ aCounter,\ aIP,\ DataKnowledge,\ CapDataMsg\rangle$

$\land$ UNCHANGED $Public\_vars$

Action 11: *AttackerSniffDataChannel*

Attacker eavesdropping data access from legal user to target server by sniffing the data channel.

Once a new data packet is captured,it will duplicate it into its current data-knowledge set.

We don't guarantee every new data packets can be captured by attacker, it only has the opportuity to get each packets.

Variables changed: $< DataKnowledge, CapDataMsg >$

Slect a new (which means unknown to attacker till now) data packets being sent from user to *FireWall*

to simulate a successful sniff.

$SelectNewDataMsg(MsgQ,\ known) \triangleq$

IF $known \neq \{\}$

THEN The aim of capturing user data access packets is to get the exposure service info about the target server so (dIP,$dPort$) is the key knowledge.

CHOOSE $S \in$ SUBSET $Seq2Set(MsgQ) : (\forall\, x \in S : (\forall\, y \in known : (x.dIP \neq y.dIP \land x.dPort \neq y.dPort)))$

ELSE

$Seq2Set(MsgQ)$

$AttackerSniffDataChannel \triangleq$

$\land\ aState =$ "Listen"

$\land\ PureChannel(FwDataChannel) \neq \langle\rangle$ pre-condition: there exists attacker unknown target server service info.

$\wedge \exists\, i \in 1\,..\,Len(PureChannel(FwDataChannel))$:
$(\forall\, x \in CapDataMsg :\ \wedge PureChannel(FwDataChannel)[i].dIP \neq x.dIP$
$\wedge PureChannel(FwDataChannel)[i].dPort \neq x.dPort$
$\wedge PureChannel(FwDataChannel)[i].Flg =$ "TCP_SYN" A new *TCP*
)
$\wedge DataKnowledge' = DataKnowledge \cup$ post-condition: attacker learned new intelligence by a successful
$SelectNewDataMsg(PureChannel(FwDataChannel),\ CapDataMsg)$
$\wedge CapDataMsg' = CapDataMsg \cup$ All the captured packets in history recorded in *Log*.
$SelectNewDataMsg(PureChannel(FwDataChannel),\ CapDataMsg)$
$\wedge sniffCount' = sniffCount + 1$ increase statistics
$\wedge$ UNCHANGED $user\_vars$
$\wedge$ UNCHANGED $sdpsvr\_vars$
$\wedge$ UNCHANGED fw_vars
$\wedge$ UNCHANGED $server\_vars$
$\wedge$ UNCHANGED $\langle aState,\ AuthKnowledge,\ \ aSession,\ aTCPLinkSet,\ CapAuthMsg,\ aCounter,\ a$
$\wedge$ UNCHANGED $Public\_vars$

Action 12: *AttackerSpoofAuth*

Attacker build and send fake SPA messages to *SDP* controller by spoofing legal user.

The making of each fake message is based on one corrsponding element in the Auth-Knowledge set, one element in the
can only be used to produce one spoof message.

The spoof message re-use the legal user's *ID* and all other fields except *SN* (*Tstamp*) field increasing to avoid anti-rep

Variables changed: $< aSession, AuthChannel, AuthKnowledge >$

make a spoof message according a captured auth knowledge

$SpoofAuthMsg(m) \triangleq$
$[MsgID \mapsto$ "SPA_AUTH",
$sIP \mapsto m.sIP,$
$sPort \mapsto m.sPort,$
$dIP \mapsto m.dIP,$
$dPort \mapsto m.dPort,$
$ClientID \mapsto m.ClientID,$
$Tstamp \mapsto m.Tstamp + 1,$ *SN* number increase

$SvrIP \mapsto m.SvrIP,$
$SvrPort \mapsto m.SvrPort,$
$HMAC \mapsto m.HMAC,$
$Type \mapsto$ "Attacker"]

$AttackerSpoofAuth \triangleq$
$\land\ AuthKnowledge \neq \{\}$ pre-condition: there exists intellicence about user's auth message learned by sniffing.
$\land\ AuthChannel' = Append(AuthChannel, SpoofAuthMsg(\text{CHOOSE } x \in AuthKnowledge : \text{TRUE}))$ send new bui
$\land\ aSession' = aSession \cup \{SpoofAuthMsg(\text{CHOOSE } x \in AuthKnowledge : \text{TRUE})\}$ new Attack session is recorded
$\land\ AuthKnowledge' = AuthKnowledge \setminus \{\text{CHOOSE } x \in AuthKnowledge : \text{TRUE}\}$ one knowledge item can be only be
$\land$ UNCHANGED $user\_vars$
$\land$ UNCHANGED $sdpsvr\_vars$
$\land$ UNCHANGED fw_vars
$\land$ UNCHANGED $server\_vars$
$\land$ UNCHANGED $\langle aState, aTCPLinkSet, sniffCount, CapAuthMsg, aCounter, aIP, DataKnowledge, CapDat$
$\land$ UNCHANGED $\langle uChannel, FwCtlChannel, FwDataChannel, aChannel, sChannel\rangle$

Action 13: $AttackerReplayAuth$

Attacker build and send fake SPA messages to SDP controller by replay legal user's message.

The making of each fake message is based on one corrsponding element in the Auth-Knowledge set, one element in the knowledge set can only be used to produce one replay message.

Variables changed: $< aSession, AuthChannel, AuthKnowledge >$

$ReplayAuthMsg(m) \triangleq$ make replay message by duplication.
$[MsgID \mapsto$ "SPA_AUTH",
$sIP \mapsto m.sIP,$
$sPort \mapsto m.sPort,$
$dIP \mapsto m.dIP,$
$dPort \mapsto m.dPort,$
$ClientID \mapsto m.ClientID,$
$Tstamp \mapsto m.Tstamp,$
$SvrIP \mapsto m.SvrIP,$
$SvrPort \mapsto m.SvrPort,$

$HMAC \quad \mapsto m.HMAC,$
$Type \quad \mapsto$ "Attacker"]

$AttackerReplayAuth \triangleq$
$\land\ AuthKnowledge \neq \{\}$ pre-condition: there exists intellicence about user's auth message learned by sniffing.
$\land\ AuthChannel' = Append(AuthChannel,\ ReplayAuthMsg(\text{CHOOSE } x \in AuthKnowledge : \text{TRUE})$
$\land\ aSession' = aSession \cup \{ReplayAuthMsg(\text{CHOOSE } x \in AuthKnowledge : \text{TRUE})\}$ new Attack sess
$\land\ AuthKnowledge' = AuthKnowledge \setminus \{\text{CHOOSE } x \in AuthKnowledge : \text{TRUE}\}$ one knowledge item
$\land$ UNCHANGED $user\_vars$
$\land$ UNCHANGED $sdpsvr\_vars$
$\land$ UNCHANGED fw_vars
$\land$ UNCHANGED $server\_vars$
$\land$ UNCHANGED $\langle aState,\ aTCPLinkSet,\ sniffCount,\ CapAuthMsg,\ aCounter,\ aIP,\ DataKnowled$
$\land$ UNCHANGED $\langle uChannel,\ FwCtlChannel,\ FwDataChannel,\ aChannel,\ sChannel\rangle$

Action 14: *AttackerBrutalAttck*

Attacker try to brutally connect the target server only by the intelligence got from user's *Auth* message.

The making of each tcp connection is based on one auth atttack session, one element in the history auth attack session can only be used to produce one brutal attack message.

Variables changed: $< aSession, AuthChannel, AuthKnowledge, FwDataChannel >$

$AttckerBulidTcpSynPktByAuthMsg(m) \triangleq$ attack try to connect target service server as a *TCP* client, send
$[sIP \quad \mapsto aIP,$
$sPort \quad \mapsto RandomPort(aCounter,\ ATTACKER\_BASEPORT),$ local port generated randomly
$dIP \quad \mapsto m.SvrIP,$ target server info directly get from previously auth message m.
$dPort \quad \mapsto m.SvrPort,$
$Flg \quad \mapsto$ "TCP_SYN",
$Type \quad \mapsto$ "Attacker"]

$Exist\_aSession4Battck \triangleq$
$\exists\, x \in aSession : (\forall\, y \in aTCPLinkSet : x.Tstamp \neq y.AuthID)$

$Get\_aSession4Battck \triangleq$ choose an historic auth attack session to make a brutal data access attack
CHOOSE $x \in aSession : (\forall\, y \in aTCPLinkSet : x.Tstamp \neq y.AuthID)$

$AttackerBrutalAttck \triangleq$

$\land\ aSession \neq \{\}$

$\land\ (\ \lor\ aTCPLinkSet = \{\}$

$\lor\ (\ \land\ aTCPLinkSet \neq \{\}$ pre-condition: there exists at least one auth attack session without brutal attack had happene

$\land\ Exist\_aSession4Battck$

$)$

$)$

$\land\ aCounter' = aCounter + 1$ acounter is used to build the local port value of the TCP connection, increase each time to a

$\land\ FwDataChannel' = Append(FwDataChannel,\ AttckerBulidTcpSynPktByAuthMsg(Get\_aSession4Battck))$

$\land\ aTCPLinkSet' = aTCPLinkSet \cup \{$ maintain local TCP socket

$[sIP \mapsto AttckerBulidTcpSynPktByAuthMsg(Get\_aSession4Battck).sIP,$

$sPort \mapsto AttckerBulidTcpSynPktByAuthMsg(Get\_aSession4Battck).sPort,$

$dIP \mapsto AttckerBulidTcpSynPktByAuthMsg(Get\_aSession4Battck).dIP,$

$dPort \mapsto AttckerBulidTcpSynPktByAuthMsg(Get\_aSession4Battck).dPort,$

$State \mapsto$ "SYN_SENT", the tcp link's state now is SYN_SENT

$AuthID \mapsto Get\_aSession4Battck.Tstamp$ this field is used to relate to the corresponding auth attack session.

$] \}$

$\land$ UNCHANGED $user\_vars$

$\land$ UNCHANGED $sdpsvr\_vars$

$\land$ UNCHANGED fw_vars

$\land$ UNCHANGED $server\_vars$

$\land$ UNCHANGED $\langle aState,\ AuthKnowledge,\ aSession,\ sniffCount,\ CapAuthMsg,\ aIP,\ DataKnowledge,\ CapDa$

$\land$ UNCHANGED $\langle uChannel,\ AuthChannel,\ FwCtlChannel,\ aChannel,\ sChannel \rangle$

Action 15: $AttackerInspectSvr$

Attacker try to connect target server according to intelligence of previously captured data plane traffic info by sending $TCP\ SYN$ pa

The making of each tcp connection is based on one element in the Data Knowledge set which is learned by sniffing legal user's data a

with target server.

one knowledge can only be used to produce one inspection attempt.

Variables changed: $< aCounter, FwDataChannel, aTCPLinkSet, DataKnowledge >$

$AttckerBulidTcpSynPktByData(p) \triangleq$

$[sIP \mapsto aIP,$

$sPort \quad \mapsto RandomPort(aCounter,\ ATTACKER\_BASEPORT),$
$dIP \quad \mapsto p.dIP,$
$dPort \quad \mapsto p.dPort,$
$Flg \quad \mapsto$ "TCP_SYN",
$Type \quad \mapsto$ "Attacker"]

$AttackerInspectSvr \triangleq$
$\land\ DataKnowledge \neq \{\}$ pre-condition: there exists learned data knowledge that still not used to makea inspe
$\land\ aCounter' = aCounter + 1$ acounter is used to build the local port value of the TCP connection, increase
$\land\ FwDataChannel' = Append(FwDataChannel,\ AttckerBulidTcpSynPktByData(\text{CHOOSE } x \in$
$\land\ aTCPLinkSet' = aTCPLinkSet \cup \{$ maintain local TCP socket
$[sIP \quad \mapsto AttckerBulidTcpSynPktByData(\text{CHOOSE } x \in DataKnowledge : \text{TRUE}).sIP,$
$sPort \quad \mapsto AttckerBulidTcpSynPktByData(\text{CHOOSE } x \in DataKnowledge : \text{TRUE}).sPort,$
$dIP \quad \mapsto AttckerBulidTcpSynPktByData(\text{CHOOSE } x \in DataKnowledge : \text{TRUE}).dIP,$
$dPort \quad \mapsto AttckerBulidTcpSynPktByData(\text{CHOOSE } x \in DataKnowledge : \text{TRUE}).dPort,$
$State \quad \mapsto$ "SYN_SENT", the tcp link's state now is SYN_SENT
$AuthID \quad \mapsto UNKNOWN\_AUTH\_ID$ This tcp connection is built accroding to captured data plane
$] \}$
$\land\ DataKnowledge' = AuthKnowledge \setminus \{\text{CHOOSE } x \in DataKnowledge : \text{TRUE}\}$ one knowledge item
$\land$ UNCHANGED $user\_vars$
$\land$ UNCHANGED $sdpsvr\_vars$
$\land$ UNCHANGED fw_vars
$\land$ UNCHANGED $server\_vars$
$\land$ UNCHANGED $\langle aState,\ AuthKnowledge,\ aSession,\ sniffCount,\ CapAuthMsg,\ aIP,\ CapDataMs$
$\land$ UNCHANGED $\langle uChannel,\ AuthChannel,\ FwCtlChannel,\ aChannel,\ sChannel\rangle$

Action 16: $AttackerRcvSynAck$

Attacker's inspection TCP connection estalished trigered by receiving $TCP\ SYN\ ACK$ pakcet from target server.

This indicates the attacker fufilled a inspection attack to the target server.

Because the Firewall dose not filter server to endpoint direction packets, so to simplify the model, the server direcly se

$uChannel$ or $aChannel$ to simplify the model.

Variables changed: $< aTCPLinkSet, aChannel, FwDataChannel >$

$AttackerRcvSynAck \triangleq$

$\wedge\ aTCPLinkSet \neq \{\}$

$\wedge\ aChannel \neq \langle\rangle$

$\wedge\ Head(aChannel).Flg \quad =$ "TCP_SYN_ACK"

$\wedge\ Head(aChannel).Type =$ "Attacker"

$\wedge\ HasMatchLink(Head(aChannel),\ aTCPLinkSet)$

$\wedge\ GetMatchLink(Head(aChannel),\ aTCPLinkSet).State =$ "SYN_SENT" pre-condition: local TCP client in the

$\wedge\ aTCPLinkSet' = (aTCPLinkSet \setminus \{GetMatchLink(Head(aChannel),\ aTCPLinkSet)\})$ Post-condition: The

$\cup\ \{[sIP \mapsto GetMatchLink(Head(aChannel),\ aTCPLinkSet).sIP,$
$sPort \mapsto GetMatchLink(Head(aChannel),\ aTCPLinkSet).sPort,$
$dIP \mapsto GetMatchLink(Head(aChannel),\ aTCPLinkSet).dIP,$
$dPort \mapsto GetMatchLink(Head(aChannel),\ aTCPLinkSet).dPort,$
$State \mapsto$ "ESTABLISHED",
$AuthID \mapsto GetMatchLink(Head(aChannel),\ aTCPLinkSet).AuthID$
$]$
$\}$

$\wedge\ aChannel' = Tail(aChannel)$

$\wedge\ FwDataChannel' = Append(FwDataChannel,\ EndPointBulidTcpAckPkt(Head(aChannel),\ \text{"Attacker"}))$

$\wedge$ UNCHANGED $user\_vars$

$\wedge$ UNCHANGED $sdpsvr\_vars$

$\wedge$ UNCHANGED fw_vars

$\wedge$ UNCHANGED $\langle aState,\ AuthKnowledge,\ \ aSession,\ sniffCount,\ CapAuthMsg,\ aCounter,\ aIP,\ DataKnowl$

$\wedge$ UNCHANGED $server\_vars$

$\wedge$ UNCHANGED $\langle uChannel,\ AuthChannel,\ FwCtlChannel,\ sChannel\rangle$

The init description of the whole system

$Init \triangleq \wedge\ UsrInit$
$\wedge\ SDPSvrInit$
$\wedge\ FwInit$
$\wedge\ AttackerInit$
$\wedge\ ServerInit$

Next state transtion of the whole system

The next state actions of the whole system is the disjunction of each entity's next state action.

$Next \triangleq$ User's next state actions

$\lor$ $UsrCommitSpaAuth$

$\lor$ $UsrConnectSvr$

$\lor$ $UsrRcvSynAck$

SDP controller's next state actions

$\lor$ $SDPSvrProcSpaAuth$

Fire Wall's next state actions

$\lor$ $FwProcAclCfg$

$\lor$ $FwProcEndPointAccess$

$\lor$ $FwProcAclTimeOut$

Attacker's next state actions

$\lor$ $AttackerSniffAuthChannel$

$\lor$ $AttackerSpoofAuth$

$\lor$ $AttackerReplayAuth$

$\lor$ $AttackerBrutalAttck$

$\lor$ $AttackerSniffDataChannel$

$\lor$ $AttackerInspectSvr$

$\lor$ $AttackerRcvSynAck$

Target service server's next state actions

$\lor$ $ServerRcvTCPSyn$

$\lor$ $ServerRcvTcpAck$

The specification of the whole system

$Spec \triangleq Init \land \Box[Next]_{vars}$

The Fair specification of the whole system

$FairSpec \triangleq$ WF means weak fairness, gurantee once the action is enabled, it will be triggered sooner or later.

$\land$ $Spec$ Use the fairness attribute to eliminate unnecessary stuttering states.

$\land \mathrm{WF}_{vars}(UsrCommitSpaAuth)$

$\land \mathrm{WF}_{vars}(SDPSvrProcSpaAuth)$

$\land \mathrm{WF}_{vars}(FwProcAclCfg)$

$\land \mathrm{WF}_{vars}(AttackerSniffAuthChannel)$

$\land \mathrm{WF}_{vars}(AttackerSpoofAuth)$

$\land \mathrm{WF}_{vars}(AttackerReplayAuth)$

$\land \mathrm{WF}_{vars}(UsrConnectSvr)$
$\land \mathrm{WF}_{vars}(FwProcEndPointAccess)$
$\land \mathrm{WF}_{vars}(FwProcAclTimeOut)$
$\land \mathrm{WF}_{vars}(ServerRcvTCPSyn)$
$\land \mathrm{WF}_{vars}(UsrRcvSynAck)$
$\land \mathrm{WF}_{vars}(ServerRcvTcpAck)$
$\land \mathrm{WF}_{vars}(AttackerBrutalAttck)$
$\land \mathrm{WF}_{vars}(AttackerSniffDataChannel)$
$\land \mathrm{WF}_{vars}(AttackerInspectSvr)$
$\land \mathrm{WF}_{vars}(AttackerRcvSynAck)$

$AuthSessionHasMatchAcl(s,\ AclSet) \triangleq$
IF $AclSet = \{\}$
THEN
FALSE
ELSE
$\exists\, r \in AclSet$:
($\land\ r.sIP = s.sIP$
$\land$ ($\lor\ r.sPort = MATCH\_ANY$
$\lor\ (r.sPort \neq MATCH\_ANY \land r.sPort = s.sPort)$
)
$\land\ r.dIP = s.dIP$
$\land\ r.dPort = s.dPort$
$\land\ r.action =$ "Accept"
)

Invariants to be verified

$DataAccessSafeLaw \triangleq$ attacker can not find target server service at anytime
$\land\ \forall\, x \in aTCPLinkSet :\ x.State \neq$ "ESTABLISHED"

$SPASafeLaw \triangleq$ attacker can not launch a successful SPA auth at anytime
$\land\ \forall\, x \in SDPSucSession :\ x.Type \neq$ "Attacker"

The temporal properties of the system to be verified

Temporal Property 1: $SPA\_AvailableProperty$

This formula asserts the availability of SPA service provided by the SDP controller

$AuthMessageMatch(m,\ n) \triangleq$ Both m and n are auth Sessions
$\land\ m.MsgID = n.MsgID$

$\wedge\ m.sIP = n.sIP$
$\wedge\ m.sPort = n.sPort$
$\wedge\ m.dIP = n.dIP$
$\wedge\ m.dPort = n.dPort$
$\wedge\ m.ClientID = n.ClientID$
$\wedge\ m.Tstamp = n.Tstamp$
$\wedge\ m.SvrIP = n.SvrIP$
$\wedge\ m.SvrPort = n.SvrPort$
$\wedge\ m.HMAC = n.HMAC$
$\wedge\ m.Type \quad = n.Type$

$SDP\_AclRuleMatch(m,\ r) \triangleq$ m is an auth Session, r is a *ACL Rule*
$\wedge\ m.sIP = r.sIP$
$\wedge\ r.sPort = MATCH\_ANY$
$\wedge\ uSvrInfo.IP = r.dIP$
$\wedge\ uSvrInfo.Port = r.dPort$
$\wedge\ r.protocol =$ "TCP"
$\wedge\ r.action =$ "Accept"

This formula asserts that the system's behavior eventually always meets the underlying propositions
1. All authentication sessions launched by legal users have been successfully processed by *SDP* controller.
2. All successfully processed *Auth* sessions recorded by *SDP* controller are sessions launched by legal users.
3. For all sucessfully authenticated sessions, the Fire wall has been configed corresponding *ACL Rule*.

$SPA\_AvailableProperty \triangleq$
$\Diamond\Box(\ \wedge \forall x \in uAuthSession : (\exists y \quad \in SDPSucSession : AuthMessageMatch(x,\ y))$ user → contro
$\wedge \forall x \in SDPSucSession : (\exists y \in uAuthSession :\ \ AuthMessageMatch(x,\ y))$ controller → u
$\wedge \forall x \in uAuthSession : (\exists y \quad \in (AclRuleSet \cup AgedRuleSet) : SDP\_AclRuleMatch(x,\ y))$
$)$

Temporal Property 2: *SPA_AntiDosProperty*
This formula asserts the Anti-Dos property of *SDP* controller,which means the controller
can always inspect and defeat spoof and replay attack.

The following formula asserts that every SPA replay attack inspected by the *SDP* controller is originated from the atta

$SPA\_AntiReplayProperty \triangleq$

IF $ReplaySession \neq \{\}$

THEN

$\forall\, x \in ReplaySession : (\exists\, y \in aSession : AuthMessageMatch(x,\, y))$

ELSE

TRUE

The following formula asserts that every SPA spoof attack inspected by the *SDP* controller is originated from the attacker

$SPA\_AntiSpoofProperty \triangleq$

IF $SpoofSession \neq \{\}$

THEN

$\forall\, x \in SpoofSession : (\exists\, y \in aSession : AuthMessageMatch(x,\, y))$

ELSE

TRUE

The following formula asserts that the system's behavior eventually always meets the underlying propositions

IF attacker ever captured legal SPA packets by sniffing,then:

1. For every captured legal SPA messages, the attacker will launch a SPA attack according to the message info.

2. Every SPA attack message launched by the attacker will be inspected and blocked by the *SDP* controller.

IF attacker never captured legal SPA packets, then no SPA attack is lanched.

$SPA\_AntiDosProperty \triangleq$

$\Diamond\Box(\ \wedge\ CapAuthMsg \subseteq uAuthSession$

$\wedge\ Cardinality(CapAuthMsg) = Cardinality(aSession)$

$\wedge$ IF $aSession \neq \{\}$

THEN

$\wedge\, \forall\, x \in aSession : (\exists\, y \in (ReplaySession \cup SpoofSession) : AuthMessageMatch(x,\, y))$

$\wedge\ SPA\_AntiReplayProperty$

$\wedge\ SPA\_AntiSpoofProperty$

ELSE

$\wedge\ CapAuthMsg = \{\}$

$\wedge\ ReplaySession = \{\}$

$\wedge\ SpoofSession = \{\}$

)

$CliSvrLinkMatch(c,\ s) \triangleq$

$\land\ c.dIP = s.sIP$

$\land\ c.sIP = s.dIP$

$\land\ c.dPort = s.sPort$

$\land\ c.sPort = s.dPort$

Temporal Property 3: *UserAccessAvailProperty*

This formula asserts the availability of the data plane service ,which means

legal user can finally access the target server except the case that 3 tuple *Acl Rule* is aged before th *TCP* connection

$UserAccessAvailProperty \triangleq$

$\Diamond\Box($ $\land$ $(\forall\, x \in uTCPLinkSet :$ $\lor$ ($\land\ x.State =$ "ESTABLISHED" *scenario1: TCP* link established, an

$\land\ \exists\, y \in sTCPLinkSet : (CliSvrLinkMatch(x,\ y) \land x.State = y$

$\land\ AclMatch4Tuple(x,\ AclRuleSet)$

)

$\lor$ ($\land\ x.State =$ "SYN_SENT" *scenario2: TCP* link half-established due

$\land\ \forall\, y \in sTCPLinkSet : \neg CliSvrLinkMatch(x,\ y)$

$\land\ AclMatch3Tuple(x,\ AgedRuleSet)$

)

)

$\land\ uTCPLinkSet \neq \{\}$

)

Temporal Property 4: *SvrHidenProperty*

This formula asserts the service hidden property of the *SDP* arhitecture. which means

finally attacker can not establish any link with the target server.

$SvrHidenProperty \triangleq$

$\Diamond\Box($ $\land$ $(\forall\, x \in sTCPLinkSet :$ $\land\ x.Type \neq$ "Attacker"

$\land\ x.State =$ "ESTABLISHED") All the established link in server side are

$\land$ $(\forall\, y \in aTCPLinkSet :$ $\land\ y.State \neq$ "ESTABLISHED") Attacker as a *TCP* client, no established

)

Temporal Property 5: *FwRuleConsistentProperty*

This formula asserts that for each successful auth session in history there exists

a corresponding 3 Tuple *Acl Rule* on FW, available or aged, vice versa.

$Get3TupleAclRuleSet(S) \triangleq$

get all the 3 *Tpule Acl* rule in history.

$$
\begin{array}{l}
\text{LET } CS[T \in \text{SUBSET } S] \triangleq \text{IF } T = \{\} \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{THEN } \{\} \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE} \\
\qquad\qquad\qquad\qquad\qquad\qquad (\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{IF } (\text{CHOOSE } x \in T : \text{TRUE}).sPort = MATCH\_ANY \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{THEN} \\
\qquad\qquad\qquad\qquad\qquad\qquad \{\text{CHOOSE } x \in T : \text{TRUE}\} \cup CS[T \setminus \{\text{CHOOSE } x \in T : \text{TRUE}\}] \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE} \\
\qquad\qquad\qquad\qquad\qquad\qquad CS[T \setminus \{\text{CHOOSE } x \in T : \text{TRUE}\}] \\
\qquad\qquad\qquad\qquad\qquad\qquad) \\
\text{IN} \quad CS[S]
\end{array}
$$

$$
\begin{array}{l}
AuthRelateAcl(s,\, R) \triangleq \\
\quad \exists\, r \in R : \land\, s.sIP = r.sIP \\
\qquad\qquad\qquad \land\, r.sPort = MATCH\_ANY \\
\qquad\qquad\qquad \land\, DeCrypt(s.SvrIP,\, Key) = r.dIP \\
\qquad\qquad\qquad \land\, DeCrypt(s.SvrPort,\, Key) = r.dPort
\end{array}
$$

$$
\begin{array}{l}
AclRelateAuth(r,\, S) \triangleq \\
\quad \exists\, s \in S : \land\, s.sIP = r.sIP \\
\qquad\qquad\qquad \land\, r.sPort = MATCH\_ANY \\
\qquad\qquad\qquad \land\, DeCrypt(s.SvrIP,\, Key) = r.dIP \\
\qquad\qquad\qquad \land\, DeCrypt(s.SvrPort,\, Key) = r.dPort
\end{array}
$$

$FwRuleConsistentProperty \triangleq$ the consistent between user's SPA session and ever configed *L3* tuple *Acl Rule* on Fire Wall

$$
\begin{array}{l}
\Diamond\Box(\, \land\, Cardinality(uAuthSession) = Cardinality(Get3TupleAclRuleSet(AclRuleSet \cup AgedRuleSet)) \\
\qquad \land\, \forall\, x \in uAuthSession : AuthRelateAcl(x,\, Get3TupleAclRuleSet(AclRuleSet \cup AgedRuleSet)) \\
\qquad \land\, \forall\, y \in Get3TupleAclRuleSet(AclRuleSet \cup AgedRuleSet) : AclRelateAuth(y,\, uAuthSession) \\
\quad)
\end{array}
$$

Temporal Property 6: *FwCorrectProperty*

This formula asserts that the Fire Wall's Packets filitering function works well, which means that for any unestablished *TCP* links there must exists packets dropping by *FireWall*.

$$
\begin{aligned}
&WithDropPkts(x) \triangleq \\
&\quad \exists\, p \in DropPackets : \ \wedge\, p.sIP = x.sIP \\
&\qquad\qquad\qquad\qquad\quad \wedge\, p.sPort = x.sPort \\
&\qquad\qquad\qquad\qquad\quad \wedge\, p.dIP = x.dIP \\
&\qquad\qquad\qquad\qquad\quad \wedge\, p.dPort = x.dPort
\end{aligned}
$$

$$
WithOutDropPkts(x) \triangleq \neg WithDropPkts(x)
$$

$$
\begin{aligned}
&FwCorrectProperty \triangleq \text{ to simplify the model, we don't consider } TCP \text{ packets re-transport mechanism, so establis} \\
&\quad \Diamond\Box(\ \wedge \forall\, x \in aTCPLinkSet : \ \text{IF } x.State = \text{“ESTABLISHED”} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{THEN} \\
&\qquad\qquad\qquad\qquad\qquad\qquad WithOutDropPkts(x) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE} \\
&\qquad\qquad\qquad\qquad\qquad\qquad WithDropPkts(x) \\
&\qquad\quad \wedge \forall\, x \in uTCPLinkSet : \ \text{IF } x.State = \text{“ESTABLISHED”} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{THEN} \\
&\qquad\qquad\qquad\qquad\qquad\qquad WithOutDropPkts(x) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE} \\
&\qquad\qquad\qquad\qquad\qquad\qquad WithDropPkts(x) \\
&\quad)
\end{aligned}
$$

$$
test2 \triangleq uState \in \{\text{“Start\_Auth”}, \text{“Auth\_End”}, \text{“Connecting”}, \text{“Connected”}\}
$$

Example:

```
Terminal ≜ ◇□(
           \ * ^
         )
```

\ * Modification History
\ * Last modified *Thu Feb* 24 15:34:13 *CST* 2022 by 10227694
\ * Created *Tue Dec* 28 09:34:21 *CST* 2021 by 10227694