Attack-Defense Trees in Secur*IT*ree®

Introduction to Attack-Defense Trees

Attack tree models have existed since the 1990s (or possibly even the late 1980s). They have proven to be extremely useful for exploring how adversaries might attack systems.

Attack trees are a bit like the *Mouse Trap* board game – a Rube Goldberg-like mouse catching device – wherein once the machine is set in motion a complex sequence of events occur that ultimately leads to the capture of a mouse. Of course, in the real world, the mouse might well notice movement in the machine and deduce that it was being attacked (before the mouse catching basket should drop). This might allow it to quickly take defensive action and protect itself by placing a stick in one of the machine's gears. A classic attack tree would not capture the dynamic interplay between the attacker (the mouse trap machine) and the defender (the mouse).

Academic researchers have recognized this shortcoming and criticized the static nature of attack trees. Various researchers have proposed extensions to the attack tree metaphor and called the result *attack-defense trees*. It should be noted that there seem to be as many different views of exactly what comprises an attack-defense tree as there are researchers. One of the most lucid descriptions is found in a 2011 paper by Roy, Kim and Trivedi, "ACT: Towards unifying the constructs of attack and defense trees", published in the Security and Communication Networks Journal.

Attack-Defense Trees in Secur*IT*ree

Overview

Support for attack-defense trees in Secur*IT*ree will begin with version 5.1 (August 2020). It should be noted that Amenaza's implementation of attack-defense trees does not attempt to mimic any particular academic paper description, although it takes inspiration from them.

Three elements are used to depict an attack-defense tree (or subtree) in Amenaza's version of attack-defense trees. The top of any attack-defense tree must always be an *AND* node. In a normal *AND* node, all of the *AND's* child nodes (or subtrees) must be completed in order for the *AND* condition to be satisfied. The difference with an attack-defense *AND* node is that some of the child operations are conditional.

Each subtree beneath the attack-defense *AND* node depicts a set of attack scenarios (or, in the case of a probability-based countermeasure, fault tree cut-sets). Secur*IT*ree models follow the convention that, if order is important, the *AND*'s children should be arranged from left to right.

The various leaf node combinations that satisfy the logic of *AND*'s subtrees correspond to paths through the subtrees. In the real world, the leaf nodes correspond to actions by the attacker and the *AND* and *OR* nodes above to states achieved by the combinations of leaf nodes. A clever defender might identify that successful attacks might require the attacker to pass through certain of these nodes. In that case, the defender might place *sensors* in the real world locations corresponding to those nodes. For instance, if a series of attacks required the adversary to pass through a door, then a sensor could be placed at the door. In the case of network attacks against critical servers, an intrusion detection system might monitor a network segment. Those attack scenarios that traverse a sensor node would cause the sensor to be *tripped* or *triggered* and the defender alerted to the presence of the attacker.

Once the defender is aware that an attack is underway, they would respond appropriately. In the case of the door alarm being triggered, a security guard might be dispatched to investigate. Note that the guard might not normally patrol that area, but the security alert would generate a special response.

That response would occur after the event was detected. The response would be depicted as either an additional attack subtree describing a sequence of events the attacker would have to overcome to defeat the additional capabilistic countermeasure (e.g., knock out the security guard) or, if the countermeasure were probabilistic in nature, hope that the countermeasure would fail of its own accord.

Both the subtree containing the sensor and the additional countermeasure subtree (whether capabilistic or probabilistic in nature) must be located directly beneath the attack-defense *AND* node, and according to our conventions of ordering activities from left to right, the response countermeasure subtree must always appear to the right of the subtree containing the sensor.

Creating an Attack-Defense Tree

To create an *attack-defense* tree (or, more usually, subtree) in Secur*I*Tree the analyst first identifies (or creates) the *AND* node that will be the attack-defense tree's parent – the *Attack-Defense AND node* (or simply, *A-D AND node*). In most regards, this *AND* node is similar to ordinary *AND* nodes in the tree. However, it differs in that it contains a list of *sensor-defense pairs* that describe the relationships between sensors and the defender's responses when a particular sensor is tripped. Consider the tree in **Figure 1** below.

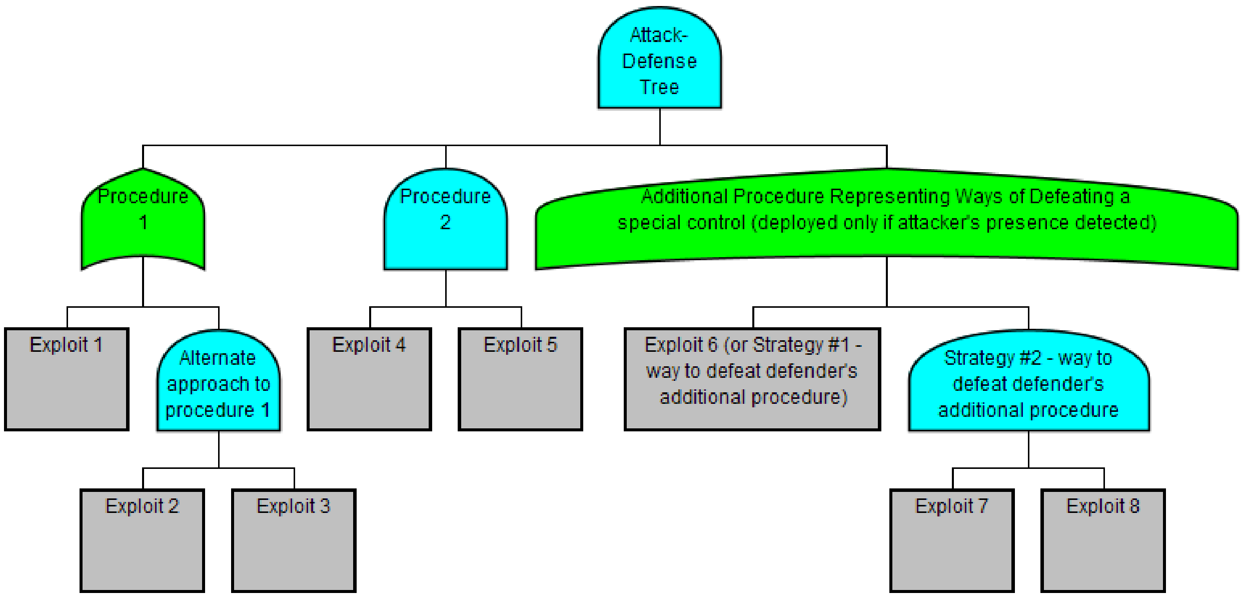

Attack-Defense Tree
Procedure 1
Procedure 2
Additional Procedure Representing Ways of Defeating a special control (deployed only if attacker's presence detected)
Exploit 1
Alternate approach to procedure 1
Exploit 4
Exploit 5
Exploit 6 (or Strategy #1 - way to defeat defender's additional procedure)
Strategy #2 - way to defeat defender's additional procedure
Exploit 2
Exploit 3
Exploit 7
Exploit 8

Figure 1 – Evolving an attack tree into an attack-defense tree

As drawn, all three of the subtrees below *Attack-Defense Tree* need to be achieved in order for the top root node to be fulfilled. But suppose that the right-most subtree, *Additional Procedure*, was expensive and that the defender didn't want to deploy it unless they felt they were under attack. Analysis might show that the adversary would be most likely to complete *Procedure 1* by using the *Alternate approach to procedure 1* (involving *Exploit 2* and *Exploit 3*). It would therefore make sense for the defender to install a sensor that would tell themself if the adversary had managed to perform the *Alternate approach to procedure 1.* The defender would respond by deploying an additional control – a control that would require the adversary to traverse the *Additional Procedure* subtree.

To define the sensor in the model the analyst would right-click on *Attack-defense tree* and select the *Define Sensor-Defense Pairs* option. A dialog would then open and they would click *Add* (**Figure 2**).

A new window would appear showing the *AD-AND* node and the subtrees beneath it. Begin by entering the *Sensor-defense pair description.*

Note that only subtrees containing nodes eligible to be *sensor* nodes are displayed. For instance, subtrees that are part of links are excluded. The analyst would then select *Alternate approach to procedure 1* and click *Select* (**Figure 3**)

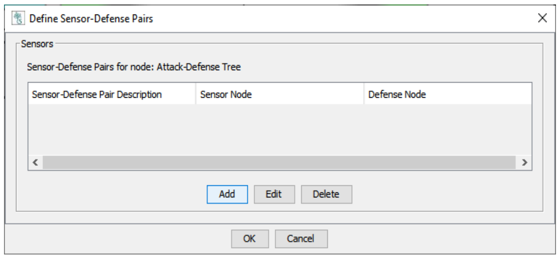

Define Sensor-Defense Pairs
Sensors
Sensor-Defense Pairs for node: Attack-Defense Tree
Sensor-Defense Pair Description
Sensor Node
Defense Node
Add
Edit
Delete
OK
Cancel

Figure 2 – Sensor-Defense Pair List Creation

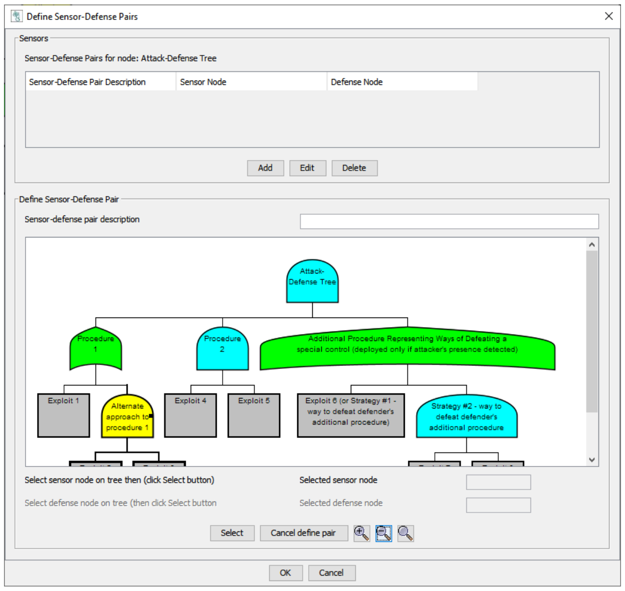
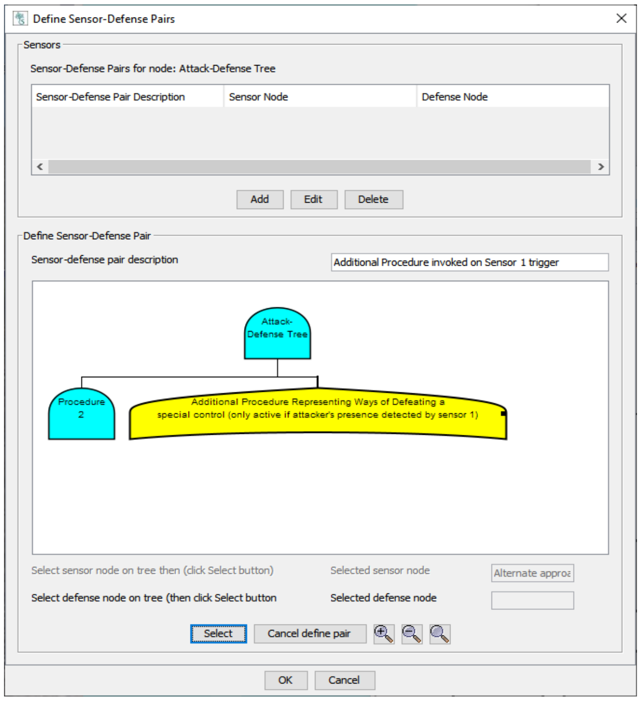

Define Sensor-Defense Pairs
Sensors
Sensor-Defense Pairs for node: Attack-Defense Tree
Sensor-Defense Pair Description
Sensor Node
Defense Node
Add
Edit
Delete
Define Sensor-Defense Pair
Sensor-defense pair description
Attack-Defense Tree
Procedure 1
Procedure 2
Additional Procedure Representing Ways of Defeating a special control (deployed only if attacker's presence detected)
Exploit 1
Alternate approach to procedure 1
Exploit 4
Exploit 5
Exploit 6 (or Strategy #1 - way to defeat defender's additional procedure)
Strategy #2 - way to defeat defender's additional procedure
Select sensor node on tree then (click Select button)
Selected sensor node
Select defense node on tree (then click Select button
Selected defense node
Select
Cancel define pair
OK
Cancel

Figure 3 – Sensor placement on a node

The analyst must now specify which countermeasure subtree will be invoked if the previously selected sensor is traversed in an attack scenario. In this example (**Figure 4**) the *Additional Procedure ...* subtree has been selected.

In order for a defense subtree to be eligible to be selected (and displayed to the user) it needs to be to the right of the sensor node and immediately beneath the *AD-AND* node.


Define Sensor-Defense Pairs

Sensors

Sensor-Defense Pairs for node: Attack-Defense Tree

Sensor-Defense Pair Description	Sensor Node	Defense Node

Define Sensor-Defense Pair

Sensor-defense pair description: Additional Procedure invoked on Sensor 1 trigger

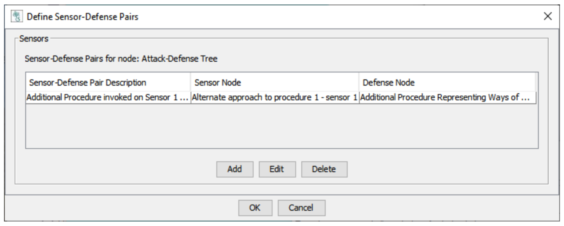
Attack-Defense Tree
Procedure 2
Additional Procedure Representing Ways of Defeating a special control (only active if attacker's presence detected by sensor 1)

Select sensor node on tree then (click Select button) — Selected sensor node: Alternate approa

Select defense node on tree (then click Select button — Selected defense node

Figure 4 – Defense subtree selection

The user then completes the creation of the sensor-pair definition in the A-D List node by clicking OK (**Figure 5**).


Define Sensor-Defense Pairs

Sensors

Sensor-Defense Pairs for node: Attack-Defense Tree

Sensor-Defense Pair Description	Sensor Node	Defense Node
Additional Procedure invoked on Sensor 1 ...	Alternate approach to procedure 1 - sensor 1	Additional Procedure Representing Ways of ...

Figure 5 – Completion of S-D pair

The completed A-D tree now appears (**Figure 6**).

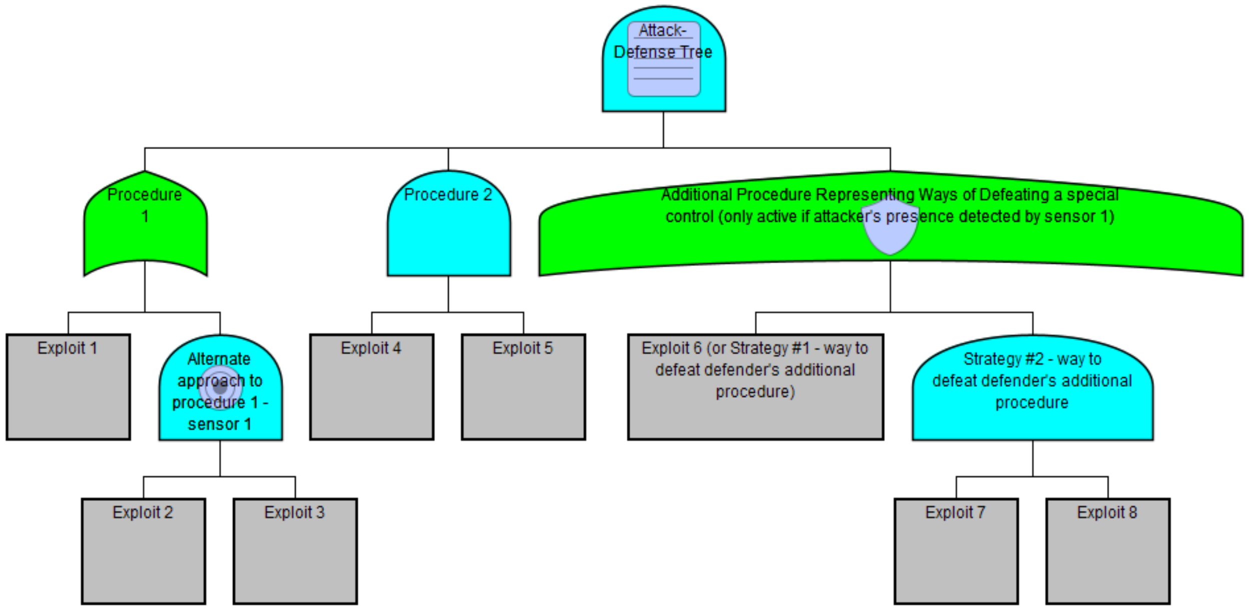
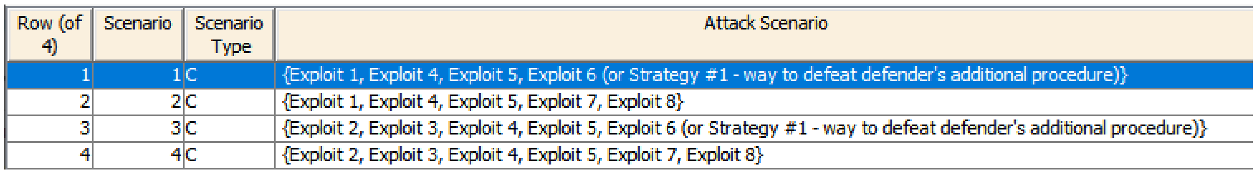

Attack-Defense Tree
Procedure 1
Procedure 2
Additional Procedure Representing Ways of Defeating a special control (only active if attacker's presence detected by sensor 1)
Exploit 1
Alternate approach to procedure 1 - sensor 1
Exploit 4
Exploit 5
Exploit 6 (or Strategy #1 - way to defeat defender's additional procedure)
Strategy #2 - way to defeat defender's additional procedure
Exploit 2
Exploit 3
Exploit 7
Exploit 8

Figure 6 – Completed Attack-Defense tree

The following are the attack scenarios (**Figure 7**).



Row (of 4)	Scenario	Scenario Type	Attack Scenario
1	1	C	{Exploit 1, Exploit 4, Exploit 5, Exploit 6 (or Strategy #1 - way to defeat defender's additional procedure)}
2	2	C	{Exploit 1, Exploit 4, Exploit 5, Exploit 7, Exploit 8}
3	3	C	{Exploit 2, Exploit 3, Exploit 4, Exploit 5, Exploit 6 (or Strategy #1 - way to defeat defender's additional procedure)}
4	4	C	{Exploit 2, Exploit 3, Exploit 4, Exploit 5, Exploit 7, Exploit 8}

Figure 7 – Attack scenario list for A-D tree